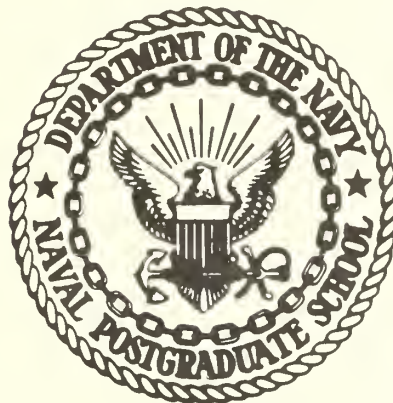


NPS-53ZZ7306-1A

United States Naval Postgraduate School



BENCHMARKED COMPARISON OF THE
TSS/360, CP/67, MTS and OS/MVT
COMPUTER OPERATING SYSTEMS

by

Gordon H. Syms

Approved for public release; distribution unlimited

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral Mason Freeman, USN
Superintendent

M. U. Clauser
Provost

BENCHMARKED COMPARISON OF THE
TSS/360, CP/67, MTS and OS/MVT
COMPUTER OPERATING SYSTEMS

ABSTRACT:

A set of terminal scripts and benchmarks have been derived for comparing the performance of time sharing and batch computer operating systems. Some of the problems encountered in designing valid benchmarks for comparing computer operating systems under both terminal and batch loads are discussed.

The results of comparing TSS/360, CP/67 and MTS time sharing systems for the IBM 360/67 over a wide range of load conditions are presented. The results of comparing TSS, MTS and OS/MVT under batch loads are also presented.

Serious performance degradation of the time sharing computer systems from overloading was experienced and a simple solution is suggested to prevent such degradation. The degradation was so severe as to render the performance less than that of a sequential job processor system.

This task was supported by an NPS Foundation Grant for two and a half academic quarters.

Acknowledgement

The author would like to express his appreciation to all the people who assisted in this project and made this report possible. Specifically, he would like to thank the following: the CS 4900 class of Quarter III AY 72 for developing the scripts and for conducting the CP and TSS benchmarking runs; William R. Haines and James H. Porterfield who performed the original analysis of the CP and TSS test results; E. E. Gregeris and R. F. Belanger who performed the terminal probing tests to determine the normal terminal user loads; Elbert F. Hinson who almost single handedly conducted the MTS tests and originally analyzed the results; the NPS computer center personnel and Gerald L. Barksdale who conducted the batch benchmarking runs and analyzed the results; and finally a special thanks to Robert E. Graham and F. Douglas Meyers who painstakingly rechecked the intermediate data and prepared many of the tables and graphs in this report.

The author would also like to thank the NPS Computer Center for providing the dedicated computer time and support for running the benchmarking runs, and also any users who were inconvenienced by the tests. Finally, he would like to express his appreciation for the NPS Foundation grants that made this project and report financially feasible.

Table of Contents

	<u>Page</u>
1 INTRODUCTION AND SUMMARY	1
1.1 Introduction	1
1.2 Operating Systems	2
1.3 Computer Configuration at NPS	4
1.4 Configurations Tested	5
1.5 Summary of Objectives	6
1.6 Summary of Results	6
2 METHODS OF APPLYING LOADS AND MEASURING PERFORMANCE	9
2.1 Possible Methods of Applying Loads	9
2.2 Actual Loads Used	11
2.2.1 Terminal Benchmarks	13
2.2.2 Batch Benchmarks	17
2.3 Possible Methods of Measuring Performance	18
2.4 Actual Measurements Used	19
3 BENCHMARK TESTS PERFORMED	21
3.1 Terminal Benchmark Tests	21
3.2 Batch Benchmark Tests	25
3.3 Other Tests	25
4 DATA COLLECTION AND ANALYSIS	29
4.1 General Philosophy	29
4.2 Throughput Analysis	30
4.2.1 Measured Throughputs	30
4.2.2 Calculated Throughputs	31
4.2.3 Measured Versus Calculated Throughputs	33
4.3 Definition of Load Factors	34
4.4 Attempted Regression Analysis	36
5 OTHER TESTS AND INTERMEDIATE RESULTS	40
5.1 Terminal Mixed Script Tests	40
5.2 Terminal Mixed Script Results	41
5.3 Terminal Probe Tests	43
5.4 Terminal Probe Results	43
6 RESULTS AND COMPARISONS	46
6.1 General Philosophy	46
6.2 CP Test Results	47
6.2.1 CP Terminal Response Times	47
6.2.2 CP Measured Throughputs	49

	<u>Page</u>
6.2.3 CP Calculated Throughputs	53
6.2.4 CP Effective Progress Rates	57
6.2.5 CP Software Monitor Results	64
6.3 TSS Test Results	70
6.3.1 TSS Terminal Response Times	70
6.3.2 TSS Throughputs	73
6.4 CP and TSS Performance Comparisons	73
6.4.1 Terminal Response Time Comparisons	73
6.4.2 Terminal Throughput Comparisons	79
6.5 MTS Test Results	84
6.5.1 MTS Terminal Response Times	84
6.5.2 MTS Throughputs	89
6.5.3 MTS Effective Progress Rates	94
6.6 CP and MTS Comparisons	98
6.6.1 CP and MTS Terminal Response Time Comparisons	100
6.6.2 CP and MTS Throughput Comparisons	104
6.6.3 CP and MTS Effective Progress Rates	104
6.7 CP, TSS and MTS Comparisons	110
6.7.1 Pair-Wise Comparisons	110
6.7.2 Performance at Typical Loading Conditions	111
6.8 Batch Benchmarking Results and Comparisons	113
6.8.1 Batch Benchmarking Results	113
6.8.2 Batch Performance Comparisons	116
6.8.3 Comparison of MTS with CP plus OS	121
7 CONCLUSIONS	124
References	129

List of Appendices

A Terminal Scripts and Programs	131
B Batch Benchmarking Programs	154
C CP Intermediate Results	159
D TSS Intermediate Results	174
E MTS Intermediate Results	177
F Batch Intermediate Results	187

List of Figures

	<u>Page</u>
1 NPS Computer Configuration	4a
2 CP Terminal Response Time for Each Script under Various Loads	48
3 CP Measured Terminal Throughput for Various Loads	50
4 CP Total Measured Throughput for Each Script	51
5 CP Total Measured Throughput for Each Script - EDIT and FORTRAN	52
6 CP Total Measured Throughput with Various Weightings	54
7 CP Mean Calculated Terminal Throughput vs Measured Terminal Throughput - FORTEX, PLISM and PLILG	55
8 Continuation of Figure 7 - EDIT and FORTRAN	56
9 CP Total Measured and Calculated Throughput	58
10 CP Effective Progress Rate per Script	60
11 CP Total Effective Progress Rates	61
12 CP Problem (CPU) Time and Paging Rates	65
13 CP Problem Time Versus Paging Rates	67
14 Problem and Supervisor Time for Runs R41 to R47	68
15 Shift in Resource Allocation during Runs R41 to R47	69
16 TSS Terminal Response Times for Various CP Load Factors	71
17 TSS Terminal Response Times for Various Combined Load Factors	72
18 TSS Terminal Throughputs - Measured and Calculated	74
19 TSS Total Measured and Calculated Throughputs per Script	75
20 CP Terminal Response Times versus Combined Load Factors	77
21 CP versus TSS Response Times	78
22 CP versus TSS Terminal Response Times for Three Configurations	80
23 CP versus TSS Calculated Terminal Throughputs	81
24 Comparison of CP and TSS Total Throughputs	83
25 MTS Response Times for Various Loads	87
26 MTS Response Times versus Combined Load Factors	88
27 MTS Measured Terminal Throughput	90
28 MTS Total Measured Throughput for Each Script	92
29 MTS Total Measured Throughput	95
30 MTS Effective Progress Rates per Script	96

	<u>Page</u>
31 MTS Total Effective Progress Rates versus MTS Load Factors	97
32 MTS Total Effective Progress Rates versus Combined Load Factors	99
33 CP versus MTS Terminal Response Times for Various MTS Loads	101
34 CP (Scaled) versus MTS Terminal Response Times for Various MTS Loads	102
35 CP (Scaled) and MTS Terminal Response Times for Various Combined Load Factors	103
36 CP (Scaled) and MTS Terminal Response Times for Equivalent Runs	105
37 CP and MTS Total Throughput Comparisons	106
38 CP and MTS Total Effective Progress Rates	108
39 Some Typical Effective Progress Rates per Script for CP and MTS	109
40 Terminal Response Times at Selected Load Conditions for CP, TSS and MTS	112
41 Total Measured Throughput for Selected Load Conditions for CP, TSS and MTS	114
42 OS Benchmarking Results	115
43 Comparison of Total Turnaround Times for Batch Benchmarks	119
E1 Time Analysis of MTS Run R63	182

List of Tables

	<u>Pages</u>
1 Comparison of Loading Methods	12
2 Summary of CP Benchmarking Runs	22
3 Summary of TSS Benchmarking Runs	24
4 Summary of MTS Benchmarking Runs	24
5 Comparison of Terminal Benchmarking Runs	26
6 Results of Mixed Script Tests and a Comparison with Fixed Script Results	42
7 Results of the Terminal Probing Tests	44
8 MTS and Combined Load Factors Corresponding to the Normal User Loads	45
9 Total Conjectured Effective Progress Rates	63
10 CP versus TSS Comparison Configurations	76
11 Breakdown of Terminal Response Times for Run R63	86
12 Hardware Monitor Results for Comparing OS/MVT Performance with 768K bytes versus 512K bytes of Core Under Normal Operating Conditions	117
B1 Batch Benchmarking Jobstream Characteristics	156
B2 Jobstream Summary by Region and CPU Time	157
B3 Jobstream C Execution and Region Used	158
C1 CP Average Response Time at Each Terminal	161
C2 CP Mean Response Time for Each Script	166
C3 CP Mean Calculated Throughput for Each Script	167
C4 CP Measured Terminal Throughput	168
C5 CP Total Measured Throughput per Script	168
C6 CP Minimum Response Times	169
C7 CP Effective Progress Rates	170
C8 CP Software Monitor Results Summary	171
C9 CP Software Monitor Results - Test 3	172
C10 CP Individual Terminal Response Times for Determining the Effect of Memory Size	173
D1 TSS Mean Response Time for Each Script	175
D2 TSS Measured Terminal Throughputs	175
D3 TSS Calculated Terminal Throughputs	175
D4 TSS Total Measured Throughputs per Script	175
D5 TSS Total Calculated Throughputs per Script	176
E1 MTS Original Response Times and Throughputs	180
E2 MTS Corrected Response Times and Throughputs	181
E3 MTS Mean Terminal Response Times Used in Graphs	184
E4 MTS Measured Throughputs per Script	185
E5 MTS Minimum Terminal Response Times	185
E6 MTS Effective Progress Rates	186
F1 MTS Batch Benchmarking Results - September 1971	188
F2 OS/MVT Batch Benchmarking Results	189
F3 Summary of Batch Benchmarking Turnaround Times	189

Section 1

INTRODUCTION AND SUMMARY

1.1 Introduction

With the increased economic pressure on most computer centers, there is an increased need for accurate comparison of different computer systems and different operating systems on a particular computer system. The problem of selecting an operating system for a large computer system is a "difficult" task, especially when there are several available operating systems. The selection is difficult because the techniques for comparing computer performance are not well developed (especially for time sharing systems), the number of variable system parameters is almost unlimited, and the operating systems have significant effect on the performance of the computer system. In fact, selecting a better operating system may even double the performance of a four million dollar computer system.

Although monitoring the computer performance under actual operating conditions is the most accurate, it is difficult to make comparisons or measure improvements in performance under these conditions. Thus there is a need for loading the computer in such a way that it is repeatable from day to day and from one operating system to another. Although the technique of using a fixed benchmark or set of programs has been used on numerous occasions to measure the performance of batch operating systems, an equivalent technique apparently has not been widely used to measure time sharing systems.

Although the problem of optimizing the performance of time sharing systems is much more difficult than for its batch counterpart, it is felt

that the benchmarking technique is also useful for time sharing systems. In a batch operating system, there are relatively few user programs competing for system resources at any one time. Thus, it is relatively simple to collect a set of benchmark programs that are representative of a "typical" load and that could be used for evaluating and comparing the computer systems. On the other hand, in a time-sharing system there are many user programs competing for resources and, at any one time, there may be many requests for a particular resource. Since there are so many combinations of requests, it is very difficult to construct a set of test programs (or scripts) that will represent a "realistic" load for comparing several time-sharing systems. Also, the time-sharing monitors (schedulers) handle the requests differently and, as a result, the same scripts present very different loads on the different systems thereby producing large differences in performance. Thus the first objective of this project was to develop a set of terminal test programs or scripts suitable for loading and comparing the performance of time sharing systems. These scripts were then to be used to compare the performance of four operating systems for the IBM 360/67 computer under a wide variety of loading conditions.

1.2 Operating Systems

The four major operating systems for the IBM 360/67 computer and the ones that were compared are:

- (1) IBM's TSS/360 - Time Sharing System, version 7 with schedule table T49 [1]*,

*Reference 1

- (2) IBM's CP/67 - Control Program/1967 developed by the Cambridge Research Center in Cambridge, Mass., version 3.0,
- (3) MTS - Michigan Terminal System developed by University of Michigan, versions 2.0 and 2.1, and
- (4) IBM's OS/MVT - Operating System/Multiprogramming with a Variable number of Tasks, Release 15/16.

In this report, these operating systems will be referred to as simply TSS, CP, MTS and OS respectively.

Three of the operating systems, TSS, CP and MTS, were designed primarily as time sharing (or terminal) systems, but TSS and MTS also have batch handling capabilities. On the other hand, OS was designed primarily as a batch operating system , although small time sharing operating systems are sometimes run under OS as one of the tasks*. Another major distinction in these systems is that TSS and MTS can operate with one or two processors (CPU), while CP and OS can operate with only a single processor.

Since both batch and terminal services are required at this installation, the system (or systems) chosen must support both requirements. For most of the time, the IBM 360/67 computer has been run as a split system with CP providing the terminal services and OS providing the batch services. For a brief four month period from September to February 1971, TSS provided both the batch and terminal services.

*Time sharing operating systems that run under OS/MVT-such as CALL/360-OS, CPS (Conversational Programming System), ITF (Interactive Terminal Facilities) ATS (Administrative Terminal Services), APL/360 and the new TSO (Time Sharing Option of OS) offer only restrictive capabilities and generally poorer performance than the general time sharing systems and therefore are not considered here.

The second objective, and the reason for the initiation of this project in January 1971, was to compare the performance of CP with that of TSS. The third objective was to compare the total performance of the split system (CP and OS) with that of TSS. The fourth objective was to compare the performance of the split system with that of the Michigan Terminal System, MTS. The performance was to be compared by placing a terminal load on CP and a batch load on OS, and then placing the same combined terminal and batch load on MTS, and measuring the resultant performances in all cases.

The fifth and final objective was to supply management with information that will assist in their decision of whether to continue running the split system (CP and OS) or to adopt MTS as the one operating system to handle both the terminal and the batch computing requirements.

The next sections will describe the NPS computer configuration and the configurations that were tested in order to satisfy the above objectives.

1.3 Computer Configuration at NPS

The IBM 360/67 at NPS has the following configuration (Figure 1):

Two	2067	central processor units (CPUs),
Three	2365	core boxes (768K bytes),
One	2314	direct access storage unit (8 disk drives),
Eight	2311	disk drives (1/2 the speed of 2314), and
One	2301	drum storage unit.

For TSS and MTS the tests were conducted using the full configuration.

4a

On the other hand, when the system was run as a split system, the resources were divided in the following manner. OS was assigned:

One	2067	CPU,
Two	2365	core boxes (512K bytes), and
One	2314	direct access storage unit;

while CP was assigned:

One	2067	CPU,
One	2365	core box (256K bytes),
Eight	2311	disk drives, and
One	2301	drum.

1.4 Configurations Tested

Since one of the objectives was to compare the performance of TSS with CP under the configurations used at NPS, the tests were conducted with both systems configured as above, thereby giving CP a serious disadvantage. In fact, CP had only one-third the core memory, the slower disks and only one processor as compared to the full system for TSS. This comparison was made to show that if CP could compete with TSS under these conditions with terminal load only, then it would be much more economical to run this installation as a split system rather than as a dual processor system under TSS. Essentially the OS batch throughput would be obtained for free.

Thus the following six configurations were tested:

- (1) TSS - Full System
(Dual processor, 768K memory, 2314 disks, drum)

- (2) CP - Single Processor, 256K Memory, 2311 Disks, Drum
- (3) CP - as above but with 512K Memory
- (4) OS - Single Processor, 512K Memory, 2314 Disks, No Drum
- (5) OS - Full system except for single processor
- (6) MTS - Full System

1.5 Summary of Objectives

Thus the objectives of this project are summarized as follows:

(1) To develop a set of terminal scripts suitable for loading and benchmarking the performance of different time sharing operating systems.

(2) To compare the performance of CP with that of TSS under the configurations at NPS (One core box, single processor and slower disks for CP; three core boxes, dual processor and faster disks for TSS).

(3) To compare the performance of the split system (CP and OS) with that of TSS with a full configuration.

(4) To compare the performance of the split system with that of MTS - the Michigan Terminal System.

(5) To supply information for management to assist in making the decision of whether to stay with the split system or to adopt a single operating system like MTS for both batch and terminal services.

1.6 Summary of Results

Before describing the details of the tests performed and the performance measured, a summary of the results are presented in this section. The major results are:

(1) A basic set of six scripts was developed for loading and comparing performance of different time sharing operating system. (Other scripts were also developed for use in other conditions such as sampling the normal operating load.)

(2) CP with a single core box (256K bytes), one processor and slower disks out-performed TSS with three core box (768K bytes), two processors and faster disks over almost all tested loading conditions.

(3) Operating the split system with CP and OS provided much better performance than running the full system under TSS, because the batch service from OS was obtained for essentially free. See result 2 above.

(4) MTS with a full system out-performed CP running with a limited system of one or two core boxes by a wide margin. In fact, the performance from MTS with two processors and three core boxes was more than three times that of CP with one core box and a single processor.*

(5) CP with two core boxes (512K bytes) had almost double the performance of CP with only one core box (256K bytes).

(6) From results 4 and 5 above, it was concluded that MTS would out-perform CP even if both systems were given the same amount of core memory and thus essentially same resources. (There was not sufficient justification to do these tests.)

(7) TSS out-performed OS on batch-only jobs by a small margin.

(8) MTS out-performed OS on batch-only jobs by a wide margin.

*The single processor restriction is a limitation of CP, not the tests performed.

Although the tests comparing CP and MTS under terminal load only and the tests comparing OS and MTS under batch load only have been completed and are reported here, the final MTS tests in which the load includes the terminal load from CP plus the batch load from OS have not yet been completed. The final testing of MTS is currently awaiting the arrival of latest distribution of MTS, namely Distribution 3.0. Thus, it is not yet possible to make a definite recommendation to management on whether to retain the split system or to adopt MTS, although there is significant evidence in favor of MTS.

The purpose of writing this report at this time is to consolidate the results from the tests already performed, and to provide motivation for completing the comparison of the performance of the split system to that of MTS. Previously reported work from which this report has drawn included Haines' and Porterfield's thesis [2] and Hinson's thesis [3]. In both cases extensive analysis and several comparisons are made beyond that reported previously.

Section 2

METHODS OF APPLYING LOADS AND MEASURING PERFORMANCE

Before discussing the actual tests performed and presenting the results, this section will discuss some of the possible methods of applying loads and methods of measuring performance of computer systems.

Even after deciding on a computer configuration, there are many choices of techniques for empirically comparing operating systems. The following selections must be made: first, the method of applying the load; second, the method of measuring performance; third, the actual test loads to be used within the selected methodology; and fourth, the actual data to be collected within the performance measurement methodology.

2.1 Possible Methods of Applying Loads

There are seven basic methods of applying loads to batch and terminal oriented computer systems. These are:

- (a) Normal User Load: the normal user load is applied under actual operating conditions. This is the most realistic load but is the most difficult to duplicate and therefore unsatisfactory for comparing performance of different operating systems.
- (b) Batch Benchmark: a set of programs are selected as "typical" jobs and run as a jobstream. This method is commonly used for measuring batch oriented computer systems.

- (c) Terminal Benchmark: a set of "typical" programs are selected as in the batch benchmark case, a set of terminal scripts are written to call these test programs in a pre-determined order, and finally, the number of each script to be run for each benchmark is selected.
- (d) Terminal Probe: A terminal script containing several "typical" programs is used to probe the response under normal operating conditions. This is a combination of methods (a) and (c) above.
- (e) Indirect Synthetic Job: A program is written, normally in a high level language, that has parameters to control the memory, processor (CPU) and input/output requirements. The same program with different parameters representing different resource requirements is run on several terminals. An example of such a job is a Fortran matrix multiplication program that changes (possibly in some random fashion) its memory, CPU and I/O requirements.
- (f) Direct Synthetic Job: a program is written, normally in assembler, that directly controls the memory, CPU, I/O and supervisor requirements. As in case (e) above, the same program with different parameters is run on several terminals. An example of such a job is A JOB written at the Naval Post-graduate School [4].
- (g) Direct load by another Computer: A mini-computer is used to generate the memory, CPU, I/O and supervisor requirements for the computer under evaluation. MITRE has developed such a Remote Terminal Emulator [5].

The criteria for selecting a loading method include repeatability, realism (accuracy in representing the real load), interference with normal operations, development effort, ease of performing the tests, and transferability to other systems. Table 1 shows the rating of each loading method against each criteria. Two comments concerning Table 1 are: (1) that the realism depends greatly on how consistent the normal user load is and how well it has been measured; and, (2) high interference means that a dedicated system is required to perform the test, such as in the benchmarking case.

A little simple arithmetic shows that there were a very large number of possible test conditions, even after all the above selections have been made. For example, since there were six operating systems configurations to be tested and seven types of loading, there were a total of 42 possible tests for each point on the load curve (independent variable). The next section will discuss how the number of possible test conditions was restricted.

2.2 Actual Loads Used

Four of the seven possible methods of loading a computer system were used in this project. A batch benchmark was used to compare TSS, OS and MTS under batch operation. A terminal benchmark was used to apply a terminal load for comparing TSS, CP and MTS under time sharing operation. A terminal probe was used to verify that ranges of loading used in the tests were realistic. (The probe was also used to isolate users who were overloading the system.) An indirect synthetic job was used to compare performance of CP and MTS at NPS (and at the University of Alberta in Edmonton, Alberta, Canada [6]).

CRITERIA FOR COMPARISON							
Loading Method	Applicability (batch or terminal)	Repeatability	Realism	Interference	Development effort	Ease of performing test	Transferability between systems
Normal user load	batch or terminal	very poor	the best	N.A.*	N.A.*	N.A.*	None
Benchmark	batch	good	medium	high	low	easy	good
Script	terminal	medium to good	fair to medium	high	medium	difficult	fair to medium
Probe	terminal	good	very good	minimal (5%)	medium	very easy	fair to good
Indirect Synthetic Job	batch or terminal	medium to good	medium	high**	low to medium	difficult**	good
Direct Synthetic Job	batch or terminal	very good	good	high**	medium to high	difficult**	fair
Direct load by mini-computer	terminal	very good	good	high	very high	medium	poor

*N.A. - Not applicable

**Could also operate synthetic jobs as probes rather than the assumed benchmarks.

Table 1: Comparison of Loading Methods

A fifth method of loading - the Direct Synthetic Job - has not been used in the evaluation tests although it was developed at NPS [4]. The sixth method - normal user load - was not used for the tests because of its poor repeatability from day to day and its poor transferability from system to system.

Since the number of configurations and loading methods was so large, it was decided to limit the type of loading on each configuration: batch and terminal benchmarks for TSS; terminal benchmarks, terminal probes and indirect synthetic jobs for CP single core-box system; terminal benchmarks for CP dual core-box system; batch benchmarks for the two OS configurations; and batch and terminal benchmarks for MTS. This still meant that nine different loading and configuration combinations were tested in this project. Even this number was too large for evaluating over a reasonable range of loads and thus the number of test loads had to be restricted on certain configurations. Even so, a total of about 60 test runs were made.

2.2.1 Terminal Benchmarks

Although the development of the terminal benchmarks was really an outgrowth of the batch benchmark concept, the terminal benchmarks are described first to be consistent with the later ordering of the performed tests. Actually this script loading approach was first suggested by Karush as the stimulus approach for benchmarking the ADEPT-50 time sharing system at Systems Development Corporation [7]. Since a standard set of terminal benchmarks apparently does not exist for comparing time sharing systems, a set of terminal scripts was developed and used as terminal benchmarks.

The problem of developing terminal benchmarks was more difficult than its batch counterpart and was divided into three subproblems:

- (1) Selecting a set of programs that were "typical" of user jobs,
- (2) developing terminal scripts to call the programs, and
- (3) assigning scripts (type and number) to each terminal in order to apply realistic benchmarking loads to the system.

The benchmark development was further hampered by the fact that reliable statistics on program utilization was not available for these time sharing systems - as is usually the case.

The programs selected for the benchmark were of three types: compilation, execution and edit programs. The compilation programs included a Fortran compilation, because that represented the major portion of terminal jobs at this installation, and a small and a large PL/I compilation. PL/I programs were chosen because they represent large complex compilations which have large working sets (30 to 40 pages) and use large amounts of CPU time, thus degrading the performance of any time sharing system.* The execution programs were represented by a computer-bound (Fortran) execution program and a page-bound (Fortran) execution program. The edit program was a simple program that performed some typical editing functions and then waited for a "simulated think time" period. The actual programs used in the benchmark were as follows:

*It is believed that the large PL/I jobs are representative of other heavy demand jobs such as assembler programs, large programs with many subroutines, etc.

- FORTTRAN - A routine to compile an average size (75 card) Fortran program.
- PLILG - A routine to compile a large (434 card) PL/I program.
- PLISM - A routine to compile a small (47 card) PL/I program.
- FORTEX - A routine to simulate a compute-bound job. It executed 1,000,000 additions in a loop and then printed a line at the terminal.
- PAGE - A routine that uses a large array and accesses a different page for each operation.
- EDIT - A routine that performs several edit type functions, such as locating a string in the program, moving the pointer up and down and typing some output to a terminal.

The major difficulty in selecting the programs was for the EDIT program, since each time sharing system handled editing differently. TSS stored the results after every edit command onto backup storage (disk or drum), while CP and MTS stored the editing results only when told to "SAVE". Since the TSS operation could not be changed, it was decided to make CP and MTS more consistent with TSS by having their edit programs retrieve and store the file after executing a few edit commands. The other problem with edit programs was in simulating a time delay to represent "think time". The basic idea was to execute edit commands for about 5 seconds and then to wait for 55 seconds for "think time". For TSS, a built-in real time clock was used to represent the 55 second delay by causing an interrupt. For CP the delay was produced manually by a terminal operator who restarted the editor every 55 seconds and allowed it to run for 5 seconds. For MTS, the delay was simulated by having the editor print an extra ten lines at the terminal. In actual fact, the MTS edit program printed 19 lines (9 extra) compared to two for CP, again, because of differences in the editors.

After the programs were selected for the terminal benchmarks, the next subproblem was to develop the proper terminal scripts for calling the programs. The major concern was whether to run fixed scripts - where each terminal script ran one and only one program -, or mixed scripts - where each script called several programs in some predetermined order. The fixed script approach was selected because it allowed better load control and data analysis. (It was later verified that the mixed scripts produced approximately the same response for the same load as the fixed scripts.)

The scripts were each written so that they would print the starting time at the terminal, call the program, print the completion time and then repeat by calling itself. This minimized the test operator intervention - limiting it to changing the scripts of certain terminals at the beginning of each run. Complete listings of the program and the terminal scripts are shown in Appendix A.

The third subproblem in defining terminal benchmarks is to assign the scripts (number and type) to each terminal. Early in the testing, it was decided to hold the number of terminals relatively constant at about 24 and to vary the load by adjusting the ratios of each type of script. Thus a test with many edits and only a few large PL/I or paging jobs would be a light load. In fact, a wide range of loads from light to very heavy loads was obtained by changing the ratio of edit scripts to compilation-plus-execution scripts from 5:1 to 1:5. The ratio 2:1 probably best represents the normal user load but the heavier loads were used to test

the system tolerance to overload. The actual number of each script for each test run is given in the next section (Section 3).

The major difficulty in selecting the number of each scripts to represent a desired load is that the different schedulers (the resource allocators in the operating systems) handle the same type of job differently. For example, MTS has a very effective strategy for handling compute-bound jobs - by assigning them a large time slice on the second processor -, and for handling "trouble-makers" who want a large number of pages and a lot of CPU time - by assigning them to the back of a low priority queue (possibly after giving them one long time slice). This seriously changes the effective load on the system, even when the terminals are running identical scripts. This is discussed further in the results section.

2.2.2 Batch Benchmarks

The batch benchmarks, that were used for comparing the batch operation of TSS, OS and MTS, consisted of fifteen programs arranged to make 3 separate benchmarks. The 15 programs were all three-step or four-step Fortran jobs with 12 of the jobs requiring less than 25 seconds of CPU time and less than 100K bytes of core memory, while the other three required less than 6 minutes and 300K bytes of memory. The characteristics of the programs are given in Appendix B. A complete listing of the programs can be obtained from the author.

The batch programs were organized in three benchmarks as follows:

- (1) Benchmark A - compilations with source listings,
- (2) Benchmark B - compilations without source listings, and

(3) Benchmark C - No compilation or listings, i.e., programs run from load modules on disk. Data (and Job Control Language) was still read from card reader.

The test method for using the batch benchmarks was to put all the card decks in the card reader with the computer idle waiting for input, start the card reader and then record the time from the first card read until all the jobs were processed and the last line printed.

2.3 Possible Methods of Measuring Performance

Two basic methods for measuring computer performance are:

(a) Primary Performance - performance as seen by the user, i.e., turnaround time, terminal response time and sometimes throughput. Primary performance is sometimes referred to as external performance.

(b) Secondary Performance - performance as observed within the computer itself, i.e., CPU utilization, channel utilization, memory utilization, paging rates, overhead, number of I/O operations, and sometimes throughput. Secondary performance is also referred to as internal performance.

There are several methods of measuring both primary and secondary performance. Primary performance can be measured by a stopwatch, by an elapsed-time clock in the computer or by users' dissatisfaction (or screams). Secondary performance can be measured by a software monitor or a hardware monitor.

The major advantages of primary performance over secondary performance is the relative ease of making the measurements and the fact that primary

performance is one of the major criteria in selecting a computer operating system. Another advantage, of somewhat lesser significance, is that the measurements are easy to make and require relatively little data analysis. For these reasons, primary performance measurements were used almost exclusively in this project.

This is not to suggest that hardware and software monitors are not important - in fact several have been used at NPS* - but for the main objective of the project, which was to assist management in selecting the best operating system, the primary performance measurements were more significant than hardware and software monitor measurements. Furthermore, considering the large number of configurations and loading methods to be evaluated, the smaller amount of data analysis was significant.

2.4 Actual Measurements Performed

As stated above, most of the measurements presented in this report were primary performance measurements. The primary performance measurements used were the terminal response times and the throughputs (or job completion rates) for terminal loads, and total completion times for batch benchmark tests. For terminal tests, a built-in elapsed-time clock was used to print the starting times and the finishing times on the terminals. The initiate-to-completion times for the batch jobs were measured with a stopwatch.

*For example, a hardware monitor by Boole and Babbage called the Measurement Engine has been used to measure OS [8,9], while the following software monitor have also been used: SIPE for TSS [10,11], MEASURE (New York State University, Stonybrook [12], and CPSNOOP (University of Alberta, Canada) for CP [13], SUPERMON, PROGLOOK and SMF (accounting data) for OS, and the Data Collection Facility for MTS.

In some cases the secondary performance such as percentage of time in problem state and paging rates were also recorded and are reported later. In some other cases software monitor measurements were made but not analyzed or included in the report.

Section 3

BENCHMARK TESTS PERFORMED

3.1 Terminal Benchmark Tests

All together about 60 terminal benchmarking test runs were made involving more than 38 hours of dedicated computer time. Of these, 38 produced useful results and are reported here. These reported runs include 22 runs for CP, 9 for TSS and 7 runs for MTS. Since each run had definite script assignments which represented a definite load (or benchmark), whenever the same script assignments were run on two different operating systems the runs were useful benchmarks for comparing the computer performance of the different operating systems. The terminal benchmarking runs involved calling the terminal scripts described previously in subsection 2.2.1.

Most of the test runs were performed on CP - primarily because of its extensive use and easy availability at NPS. The 22 CP test runs (plus 3 preliminary runs) are shown in Table 2 - which shows the script assignment for all runs. Runs R11, R12 and R13 were preliminary runs used to develop the scripts and the benchmarking procedures. Since they did not produce any useful results, they were usually excluded from the analysis in the later sections.

The runs in Test 2 (those beginning with the digit 2) were conducted with approximately a 2:1 ratio of edit to compilation-plus-execution scripts and with a variable number of terminals. These runs were intended as a realistic representation of the actual terminal user loads experienced

TABLE 2: SUMMARY OF CP BENCHMARK RUNS

	R11	R12	R13	R21	R22	R23	R24	R25	R31	R32	R33	R34	R35	R41	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT	2	7	7	8	10	12	14	15	4	12	10	8	6	14	13	10	7	0	0	13	15	0	0	15
PAGE	0	0	0	0	0	0	0	0	0	0	2	4	6	0	0	0	0	0	0	0	0	2	1	0
FORTEX	1	4	4	3	2	3	3	4	6	2	2	2	2	5	6	6	6	5	4	6	5	5	5	5
PLISM	0	0	0	0	0	0	0	1	3	3	3	3	3	1	1	1	1	1	2	1	1	1	1	1
PLILG	1	3	2	0	0	1	1	1	4	2	2	2	2	1	1	4	7	14	14	1	1	14	15	1
FORTTRAN	0	0	1	1	1	1	3	1	7	5	5	5	5	1	1	1	1	2	2	1	1	1	1	1
TOTAL	4	14	14	12	13	17	21	22	24	24	24	24	24	22	22	22	22	22	22	22	23	23	23	23
CP LOAD FACTOR*	16	54	50	32	32	48	60	64	110	86	130	174	218	66	68	86	104	146	148	68	68	190	174	68
COMBINED * LOAD FACTOR	16	55	49	28	24	40	46	55	204	68	36	104	122	60	65	86	107	152	152	65	61	170	168	61

*See Section 4 for calculation of Load Factor

under normal operations. The runs in Test 3 were conducted with much lower ratio of edit to compilation-plus-execution jobs than Test 2 - as low as 1:4 - with a constant 24 terminals. These runs represented much heavier loads than those in Test 2 and were designed for comparing the computer performance under heavy overload conditions. Test 4 was designed for testing the performance in the range between Tests 2 and 3, and for determining the overload recovery time. Test 5 represents another rather heavy load similar to Test 4. It also included some mixed script tests to be reported later.

The 9 TSS runs were designed to be the same as the runs in Tests 2 and 3 for CP. Since the runs are almost identical the same run identification numbers were used as shown in Table 3. Table 3 also shows the terminal script assignments for all the TSS runs. More tests were planned for TSS but after analyzing the results from these tests it was decided to discontinue the TSS evaluation.

The 7 MTS runs were designed to cover the complete range of the CP runs, providing the response time remained below the predetermined maximum limit of 20 minutes for a PLILG. Table 4 shows the terminal script assignment for the MTS runs. The MTS runs actually covered the complete spectrum of CP tests, but unfortunately, due to an error in the terminal assignments during the test, the MTS runs did not match the CP runs exactly. Therefore the closest CP run and the degree of closeness are also shown in Table 4.

TABLE 3: SUMMARY OF TSS BENCHMARK RUNS

Run ID Script	R21	R23	R24	R25	R31	R32	R33	R34	R35
EDIT	8	12	14	16	4	12	10	8	6
PAGE	0	0	0	0	0	0	2	4	6
FORTEX	2	3	3	4	6	2	2	2	2
PLISM	0	1	0	1	3	3	3	3	3
PLILG	1	1	1	0	4	2	2	2	2
FORTTRAN	1	1	3	3	7	5	5	5	5
Total	12	18	21	24	24	24	24	24	24

TABLE 4: SUMMARY OF MTS BENCHMARK RUNS

Run ID Script	R61	R62	R63	R64	R65	R66	R67
EDIT	11	13	10	8	8	5	12
PAGE	0	0	0	0	2	0	0
FORTEX	3	4	4	4	3	6	0
PLISM	0	1	1	1	3	3	1
PLILG	1	4	7	4	4	4	7
FORTTRAN	1	1	1	1	3	5	3
TOTAL	16	23	23	18	23	23	23
MTS LOAD FACTOR*	44	84	142	84	97	132	113
CLOSEST CP TEST	R23	R43	R44	R42	R33	R31	R44
DEGREE OF CLOSENESS TO CP TEST	Good	Good	Close	Poor	Poor	Good	Close

*See Section 4 for calculation of Load Factor

Unfortunately there were a limited number of runs on which direct performance comparisons can be made. There were 9 runs from Tests 2 and 3 for comparing CP and TSS. There were 5 reasonably good comparison runs for CP and MTS, although there were some minor differences in the script assignments. These differences were insignificant compared to the difference in effective load and resulting difference in performance. There were only 3 runs for comparing all three systems, but that need not hamper the comparisons of pairs of systems and the inferred comparison of all three systems.

Table 5 shows all the terminal benchmark runs with similar runs grouped together for easy comparison of minor differences in the benchmarks.

3.2 Batch Benchmark Tests

The batch benchmarking tests consisted of running the three batch benchmarks described in Section 2.2.2 on the three operating systems with batch handling capabilities - TSS, OS, and MTS. The TSS evaluation tests were conducted with the full duplex (2 processors) and the half duplex (single processor) configurations, both with three core boxes, and then with the half duplex configuration with two core boxes. OS/MVT was evaluated with a single processor configuration with two and three core boxes. (Duplex operation is impossible under OS.) MTS was tested under all three batch benchmarks with duplex, three-core box configuration, but only under Benchmark C with the single processor, two-core box system.

3.3 Other Tests

The indirect synthetic job, called CPTEST, has been used at NPS for some evaluation of CP under normal operating conditions but has not been used for terminal benchmarking. However the same program has been used

TABLE 5: COMPARISON OF TERMINAL BENCHMARKING RUNS

Run Number	Operating System	TERMINAL SCRIPTS						Total Terminals	Load Factor**
		EDIT	FORTEx	FORTran	PLISM	PLILG	PAGE		
R21*	TSS	8	2	1	0	1	0	12	28
	CP	8	3	1	0	0	0	12	32
R22*	TSS								
	CP	10	2	1	0	0	0	13	32
R23	TSS	12	3	1	1	1	0	18	40
	CP	12	3	1	0	1	0	17	48
R61	MTS	11	3	1	0	1	0	16	44
R24*	TSS	14	3	3	0	1	0	21	46
	CP	14	3	3	0	1	0	21	60
R25*	TSS	16	4	3	1	0	0	24	55
	CP	15	4	1	1	1	0	22	64
R31	TSS	4	6	7	3	4	0	24	104
	CP	4	6	7	3	4	0	24	110
R66	MTS	5	6	5	3	4	0	23	132
R32*	TSS	12	2	5	3	2	0	24	68
	CP	12	2	5	3	2	0	24	86
R33	TSS	10	2	5	3	2	2	24	86
	CP	10	2	5	3	2	2	24	130
R65	MTS	8	3	3	3	4	2	23	97
R34	TSS	8	2	5	3	2	4	24	104
	CP	8	2	5	3	2	4	24	174
R35	TSS	6	2	5	3	2	6	24	122
	CP	6	2	5	3	2	6	24	218
R36	CP***	4	6	7	3	4	0	24	110

*No equivalent MTS run

**See Section 4 for calculation of Load Factors. CP Load Factor for CP, Combined Load Factor for TSS and MTS Load Factor for MTS.

***CP with 2 core boxes. Same benchmark as Run R31. (All other CP runs with 1 core box.)

TABLE 5: COMPARISON OF TERMINAL BENCHMARKING RUNS (Continued)

Run Number	Operating System	TERMINAL SCRIPTS						Total Terminals	Load Factor**
		EDIT	FORTEX	FORTRAN	PLISM	PLILG	PAGE		
R41	CP	14	5	1	1	1	0	22	66
R42	CP	13	6	1	1	1	0	22	68
R64	MTS	8	4	1	1	4	0	18	84
R43	CP	10	6	1	1	4	0	22	86
R62	MTS	13	4	1	1	4	0	23	84
R44	CP	7	6	1	1	7	0	22	104
R63	MTS	10	4	1	1	7	0	23	142
R67	MTS	12	0	3	1	7	0	23	113
R45	CP	0	5	2	1	14	0	22	146
R46	CP	0	4	2	2	14	0	22	148
R47	CP	13	6	1	1	1	0	22	68
R51	CP	15	5	1	1	1	0	23	68
R52	CP	0	5	1	1	14	2	23	190
R53	CP	0	5	1	1	15	1	23	174
R54	CP	15	5	1	1	1	0	23	68
Load Factor	CP	2	4	4	6	8	24		
Weights**	MTS	1	8+	1	6	8+	6		
	Combined	1	6	2	6	8	10		

+Weights for ≤4 jobs; weight = 20 for the number of jobs above 4.

extensively at the University of Alberta for benchmarking the performance of CP and MTS [6].

Two other types of tests were conducted involving the use of a terminal probe for measuring the normal terminal response under actual operating conditions, and involving the running of mixed scripts to verify that they produced approximately the same terminal responses as the fixed scripts used for benchmarking. These tests, along with the results, are described in Section 5.

Section 4

DATA COLLECTION AND ANALYSIS

4.1 General Philosophy

Most of the data collected and analyzed in this project was the primary performance measurements as discussed previously in Sections 2.3 and 2.4. For the terminal benchmarking runs, the primary performance measurements included the terminal response times and the throughputs or job completion rates. For the batch benchmarking runs, the primary performance measurements were the total elapsed job completion times.

The terminal response times were collected by having the terminal scripts print the real time (wall clock time) at the commencement and completion of the programs called by the scripts. The terminal response times were extracted manually from the terminal printouts. From these figures the mean response time for each terminal was determined. These mean terminal response times, organized by script, are shown in Appendix C. The mean response time per script was determined by averaging the mean terminal response times for each script. These results were used to prepare many of the TSS and MTS graphs presented in Section 6. For CP mean terminal response times were used instead of mean response times per script.

The job completion rates, or throughputs were determined by counting the number of program completions and dividing by the length of the run. The actual throughput calculations are discussed further in the next section.

For the batch benchmarking runs, the primary performance was measured by recording the starting time for reading of the first card and the completion time for printing the last line on the line printer. In some

cases other times were also determined, such as, the elapsed time for completion of individual jobs or the elapsed time while the system was fully loaded. These times were determined from the operator's console log or from the accounting information appearing on the printout.

4.2 Throughput Analysis

Two types of throughput are defined in this section: measured throughput and calculated throughput. The measured throughput was determined by observing the number of job (or program) completions during the run and dividing that figure by the run duration. The calculated throughput is determined by the reciprocal of the mean terminal response time for that script.

4.2.1 Measured Throughput

The measured throughput was determined by counting the number of job completions of a given type within a test run and dividing by the total elapsed time of the test. This measured throughput was recorded in jobs per minute per terminal for each job type (script) for each run.

The formula for determining the measured throughput on a terminal was:

$$TP_{ij}^m = SS_{ij}/RD \quad (1)$$

where TP_{ij}^m = measured throughput of script i
on terminal j

SS_{ij} = Number of script i
that completed on terminal j ,
i.e., the sample size

RD = Run duration

The total throughput for all terminals running script i was:

$$TP_i^m = \frac{1}{RD} \sum_{j=1}^{NT_i} SS_{ij} , \quad (2)$$

where NT_i = number of terminals running script i.

The mean throughput of all terminals running script i was simply:

$$TP_i^m = \frac{1}{RD \times NT_i} \sum_{j=1}^{NT_i} SS_{ij} . \quad (3)$$

On the other hand, by counting all the jobs that completed for script i regardless of the terminal (called SS_i), the mean measured throughput formula for script i was reduced to

$$TP_i^m = SS_i / (RD \times NT_i) . \quad (4)$$

Note that jobs that were completed before the test started or jobs that had not completed by the end of the run were generally* not included in the sample size.

The total measured throughput was calculated by:

$$TP^m = \sum_{i=1}^6 TP_i^m \quad (5)$$

4.2.2 Calculated Throughput

On the other hand, the calculated terminal throughput per minute was defined as the reciprocal of the mean response time for each script:

* In a few cases where the sample size was small (i.e., very heavy loads most of the job completion time was between the run start and stop times, the job was included in the sample size.

$$TP_{ij}^c = 1/RT_{ij} , \quad (6)$$

where RT_{ij} was the mean response time for script i on terminal j .

The calculated throughput for each script (regardless of terminal) was:

$$TP_i^c = \sum_{j=1}^{NT_i} \frac{1}{RT_{ij}} , \quad (7)$$

where NT_i = the number of terminal running script i . When the response time for each script on each terminal, RT_{ij} , was replaced with the mean response time of all terminals running script i , RT_i , given by:

$$RT_i = \frac{1}{NT_i} \sum_{j=1}^{NT_i} RT_{ij} , \quad (8)$$

then the mean calculated throughput for each script was calculated by:

$$\overline{TP}_i^c = 1/RT_i . \quad (9)$$

This was actually the formula used in determining the calculated terminal throughput.

The total calculated throughput was determined by

$$TP^c = \sum_{i=1}^6 NT_i \times \overline{TP}_i^c \quad (10)$$

An alternate method for determining the total calculated throughput per script and the total calculate throughput would be:

$$TP_i^c = \sum_{j=1}^{NT_i} \frac{1}{RT_i} = \frac{NT_i}{RT_i} ,$$

or

$$TP_i^c = \frac{NT_i}{\frac{1}{NT_i} \sum_{j=1}^i RT_{ij}} = \frac{(NT_i)^2}{\sum_{j=1}^i RT_{ij}}$$

The total calculated throughput, TP^c , would be calculated by:

$$TP^c = \sum_{i=1}^6 TP_i^c .$$

This equation is similar to Equation 5 for the total measured throughput, but it was not used in the actual calculations.

4.2.3 Measured verses Calculated Throughput

When the two throughputs were compared, the calculated throughput was always higher than the measured throughput because*:

- 1) it neglected the partially completed scripts at the beginning and the end of the run, and
- 2) it neglected the overhead in initiating and terminating the scripts.

The first factor improved the accuracy of the calculated throughput over the measured throughput but the second factor had the opposite effect. Especially at heavy loads, the time to initiate and terminate a script seriously reduced the total number of completions per minute. Thus the actual throughput was between the measured and the calculated throughputs.

* In a few cases minor recording errors reversed this theoretical results.

4.3 Definition of Load Factors

In order to present the data in organized fashion, it was necessary to calibrate the loads presented by the terminal benchmarks. Since the number of terminals was not a good indication of the load, it was necessary to develop a new method of calibrating the loads. In fact, even with a constant number of terminals, the load obviously changed significantly since the terminal response times changed over a range of four to one. These requirements provide the motivation for defining a load factor.

Therefore the load factor was defined as:

$$\begin{aligned} LF = W_1 N_{EDIT} + W_2 N_{FORTRAN} + W_3 N_{FORTEX} \\ + W_4 N_{PLISM} + W_5 N_{PLILG} + W_6 N_{PAGE} \end{aligned} \quad (11)$$

or more simply as:

$$LF = \sum_{i=1}^6 W_i N_i, \quad (12)$$

where N_i is the number of script i being run (i.e., same as number of terminal running script i , i.e., NT_i in the previous section) and W_i is the weight assigned to script i . The value of the weight depends on the loading effect of the script; if W_i is low, that script applies a small load to the system but if W_i is high, that script applies a heavy load. The major problem with the load factor concept is in the determination of appropriate weights.

Since the purpose of the load factor was to present the data in an organized fashion, the load factor weights were initially determined by subjective analysis of the terminal response times and probably loading effects of each type of script used in the benchmarks. A set of weights

were arbitrarily chosen and the terminal response times were plotted for each script. Then the weights were adjusted to make all the response time graphs approximate linear functions of the load factor. This was repeated as necessary until the following load factor equations were obtained. The CP load factor (as determined from Figure 2) was:

$$\begin{aligned}
 LF_{CP} = & 2 N_{EDIT} + 4 N_{FORTRAN} + 4 N_{FORTEX} \\
 & + 6 N_{PLISM} + 8 N_{PLILG} + 24 N_{PAGE}
 \end{aligned}
 \tag{13}$$

The MTS load factor (as determined from Figure 23) was:

$$\begin{aligned}
 LT_{MTS} = & N_{EDIT} + N_{FORTRAN} + 6 N_{PLISM} + 6 N_{PAGE} \\
 & + 8 ((N_{FORTEX} \leq 4) + 2.5 (N_{FORTEX} > 4)) \\
 & + 8 ((N_{PLILG} \leq 4) + 2.5 (N_{PLILG} > 4))
 \end{aligned}
 \tag{14}$$

The extra complexity of the MTS load factor was the result of the sharp increase in the response time with more than 4 jobs of either FORTEX or PLILG scripts.

Although these load factors work very well in presenting the throughput data as well as the terminal response data for the individual systems, they do not work well for comparing the systems. In fact it is obvious that identical terminal loads produce different load factors. Therefore it was necessary to define a third load factor somewhere between the two load factors above. The chosen combined load factor was given by:

$$\begin{aligned}
 LF_{COMB} = & N_{EDIT} + 2 N_{FORTRAN} + 6 N_{FORTEX} \\
 & + 6 N_{PLISM} + 8 N_{PLILG} + 10 N_{PAGE}
 \end{aligned}
 \tag{15}$$

Some interesting results were obtained when the terminal response of one system was plotted against the load factor of another system, which provided some useful insight into the nature of the load factors. When TSS terminal response was plotted against the CP load factor the result was a concave-downward curve as the load factor increased instead of the expected linear relationship. This meant the CP load factor weights were too high for TSS, which meant the CP response was degraded more rapidly the TSS response as the load was increased. On the other hand, when TSS response time was plotted against the combined load factor it produced a linear response relationship. Furthermore when the CP response time was plotted against the combined load factor, the resultant response time curve was a concave-upward function of the load. Thus the load factor actually indicates the load experienced by the computer system as the result of that benchmark. More details of the response times, throughputs and load factors will be presented in Section 6.

4.4 Attempted Regression Analysis

Since the load factors were determined on a rather arbitrary empirical basis, an attempt was made to put the load factor determination on a more firm mathematical basis by using multiple linear regression analysis. Unfortunately this attempt was not successful due to the indirect relationships between the dependent and the independent variables, as explained below. This problem is presented here as an interesting challenge to the reader.

The load factor as given in Equation 12 was:

$$LF = \sum_{i=1}^6 W_i N_i \quad (12 \text{ repeated})$$

where N_i is the number of terminal running script i and W_i is the load factor weight assigned to script i . It is assumed that the theoretical terminal response time for each script is some function of the terminal load, say

$$RT_i^t = f(N_i) \quad (15)$$

or $RT_i^t = f(N_{\text{EDIT}}, N_{\text{FORTRAN}}, N_{\text{FORTEX}}, N_{\text{PLISM}}, N_{\text{PLILG}}, N_{\text{PAGE}})$.

The function f could be determined by regression analysis but this was not done. (Actually a linear regression was tried but the required curvi-linear regression was not attempted.) In any case this is not the problem that is of interest.

The main problem is to mathematically relate the terminal response times to the load factors. Assuming for the time being that a linear relationship exists - which is a reasonable assumption considering the definition of the load factor -, then the theoretical response time for script i is:

$$RT_i^t = a_i + b_i \text{ LF}, \quad (16)$$

where a_i is the intercept of the response time curve for script i and b_i is the corresponding slope; and LF is the load factor, or independent variable. Combining Equations 12 and 16 and changing the indices in Equation 12 from i to j produces:

$$RT_i^t = a_i + b_i \sum_{j=1}^6 W_j N_j. \quad \text{for } i = 1 \text{ to } 6 \quad (17)$$

The error between the theoretical and observed response time, RT_i^o , is given by:

$$e_i = RT_i^t - RT_i^o \quad (18)$$

Actually, there is an error for every observed response time (i.e., every benchmark), RT_{ik}^o , which is given by:

$$\begin{aligned} e_{ik} &= RT_i^t - RT_{ik}^o \\ &= a_i + b_i \sum_{j=1}^6 N_{jk} W_j - RT_{ik}^o, \end{aligned} \quad (19)$$

where N_{jk} is the number of terminals running script j on run k .

The total error E is given by:

$$E = \sum_{k=1}^{NR} \sum_{i=1}^6 a_i + b_i \sum_{j=1}^6 N_{jk} W_j - RT_{ik}^o, \quad (20)$$

where NR is the number of runs.

From the CP data there are 22 runs, or sample points, for the observed response time, i.e., $k=1$ to 22, for each of the 6 scripts making a total of 132 sample points. (Actually there are only about half this number because of the holes in the observed data.) The problem has now been reduced to finding the best values of a_i , b_i and W_j (which is the same as W_i) which minimizes the total error E . Since N_{jk} is the number of scripts j on run k and is known, there are 18 unknowns and about 65 observations. The major complication in minimizing the error is that the b 's and the W 's appear as the product in the equation.

One method that appears feasible is to replace the $b_i W_j$ product terms by another variable c_m where $m = 1$ to 36. Now the problem is a linear one with 6 unknown variables, a_i , and 36 unknown variables, c_m , and 65 known observations. The follow-on problem would be to separate the c_m variable in product components $b_i W_j$ in order to get the slope of the response time graph and the load factor weights. This is a linear programming problem with 36 equations and 12 unknowns and therefore is relatively simple.

If the reader has any suggestions on how to solve this problem, the author would appreciate hearing about them. In the meantime, the report will proceed using the rather arbitrary load factors described earlier.

Section 5

OTHER TESTS AND INTERMEDIATE RESULTS

5.1 Terminal Mixed Script Tests.

For part of the Test 5 described earlier, mixed scripts were run to confirm the hypothesis that the fixed scripts used in the benchmark tests were a realistic method of loading the computer under evaluation. This test was designed to confirm that mixed scripts produced essentially the same performance as fixed scripts and that the fixed scripts were easier to control and conduct data analysis on. An intermediate load was chosen (Run R44) and three mixed scripts were designed to compare the performance with Test 4 results.

Three scripts were chosen to avoid the cycling problem whereby the same type of programs had a tendency to become synchronized in their request for resources. For example a program that required a lot of processor time while other programs were requiring channel and drum resources for paging would run very quickly (no one else wanted the CPU) and soon get to a part of its script that required a lot of pages; and then it would have to compete with all the other jobs for channel and drum resources. It would then be synchronized with the other jobs.

The same number and type of programs were included in all three scripts, but the order of execution was different. The precise scripts are shown in Appendix A as ALL1, ALL2 and ALL3.

5.2 Terminal Mixed Script Results

Since fixed scripts were easier to control, produced results that were easier to analyze, and made computer performances easier to compare, it was conjectured that fixed scripts could be used in place of the more general mixed scripts. Table 6 shows a comparison of the response to four programs during fixed scripts runs and mixed script runs. Although the mean values of the responses are very close to those of the run from which the mixed scripts were derived (Run 44), there was a lot of variation in the mixed script responses. Sometimes the minimum response was about one-third of the mean value while the maximum was 3 times the mean response. In fact the minimum response for mixed scripts is quite close to run R42, while the maximum responses indicate the load was higher than any used in the benchmarking runs (see Table 6). The fixed script responses are taken from Table C2 in Appendix C.

The load factor that corresponds to the mixed script responses is also shown in Table 6. The minimum and mean responses correspond to load factors of 68 and 106 respectively while the maximum responses represent a load factor of over 200. These correspond very well to the load factors for R42, R44 and R34. The load factors were determined from the terminal response curves to be shown in Figure 2 in Section 6.

The above evidence justified the use of fixed scripts for benchmarking.

The evidence showed that the load variance within a run using fixed scripts was much less than for mixed scripts, and that both types of scripts, produced approximately the same mean response times.

Table 6 Results of Mixed Script Tests
and a Comparison with Fixed
Script Results

Script	Mixed Script Response (minutes)			Fixed Script Mean Responses* (minutes)			Load Factor for Mixed Script Responses**		
	Min.	Mean	Max.	R42	R44	R34	Min.	Mean	Max.
FORTRAN	1.4	3.5	7.2	1.27	2.40	5.17	65	105	>200
FORTEX	4.0	9.7	23.0	5.09	10.5	--	60	100	>200
PLISM	4.3	10.2	27.6	3.63	5.85	14.4	65	105	>200
PLILG	12.2	20.8	45.6	11.4	19.9	40.6	80	115	190
Average CP Load Factor				68	104	174	68	106	>200
Percentage Problem time		35		42	29	7.6			
Page reading rate (per min.)		20.7		23.5	20	32			

* Fixed Script responses are obtained from Table C2 in Appendix C.

**The load factors are taken from the terminal response curves to be shown in Figure 2 in Section 6.

5.3 Terminal Probe Tests

The purpose of the terminal probe tests was to confirm that the benchmark loads were a realistic representation of the actual terminal loads under normal operating conditions. The terminal probe tests were designed to sample the actual response under normal operating conditions. Since the terminal probe was only one of many (7 to 21) normal terminal users and since its effective load was roughly known from the mixed script runs, the effect of the terminal probe on measured performance was to add another "rather typical" user.

A total of 25 terminal probe runs were made on CP on a daily basis over the period April 28, 1971 to June 4, 1971. The terminal probe script is a combination of the same six programs used in the terminal benchmark (See section 2.2.1). The exact script is shown in Appendix A under the name MIX.

5.4 Terminal Probe Results

The terminal probe tests confirmed that the terminal benchmarks represented realistic terminal loads. As conjectured, the actual computer load, as measured under normal operating conditions by the terminal probe tests, was approximately equivalent to runs R21 to R25. At times the measured load was almost as high as the medium heavy loads like R32, R41, R42, R43, etc, but it was never as high as that applied by the heavy overloads on some of benchmark runs.

The results of the 25 daily samples of the normal load are summarized in Table 7. The mean response times were: for FORTRAN, 0.42 minutes; for PLISM, 3.4 minutes; and for PLILG, 4.5 minutes. The corresponding maximum

Table 7 Results from Terminal Probing Tests

Script	Response Times (minutes)				Corresponding Load Factors from Figure 2 Section 6		
	Minimum	Mean	Maximum	Standard Deviation	Minimum	Mean	Maximum
EDIT	.033	.19	1.0	.27	<30	40	>70
FORTRAN	.12	.42	1.37	0.39	<40	52	65
FORTEX	.38	1.28	4.1	1.04	20	45	62
PLISM	.5	3.4	11.1	2.7	25	55	110
PLILG	1.15	4.5	15.0	2.9	20	50	90
PAGE	1.57	4.4	11.1	2.4	--	--	--
2nd EDIT	.033	.26	.97	.25	<30	50	>70
TOTAL	6.8	16.2	36.2	7.2			
PAGE (scaled)*	6.5	18.1	45.8	9.6			
TOTAL (scaled)*	11.9	29.9	71.7	--			
Mean Load Factor					≤25	50	80

*Scaling because main loop in PAGE was decreased by a factor of 4.1 from 1030 to 250 in order to reduce the total response time. (Size of matrices stayed the same.)

response times were: for FORTRAN, 1.4 minutes; for PLISM, 11.1 minutes; and for PLILG, 15.0 minutes. The load factors corresponding to the average actual computer load and to the maximum sampled actual load are 50 and 80 respectively. (These load factors were determined from Figure 2 in Section 6.) Since the benchmarking tests had load factors of 30 to 200, the benchmarking runs do, in fact, represent normal, heavy and very heavily overloaded conditions.

In order to determine values for the combined load factor and the MTS load factor that were representative of the normal user loads and the heaviest sampled user loads, a plot was made of combined load factor and MTS load factor versus CP load factors for all the benchmarking runs. From this graph, the corresponding values of 45 and 46 for the normal user load and 70 and 74 for the heaviest sampled load were obtained for the combined load factors and the MTS load factors, respectively. These values are summarized in Table 8.

Table 8 MTS and Combined Load Factors
Corresponding to the Normal
User Loads

		Average Sampled User Load	Heaviest Sampled User Load
CP Load Factor		50	80
Corresponding Combined Load Factor	Range	38 to 52	57 to 84
	Mean	45	70
Corresponding MTS Load Factor		46	74

Section 6

RESULTS AND COMPARISONS

6.1 General Philosophy

This section presents the results of more than 38 terminal benchmarking runs and 22 batch benchmarking runs, together with the performance comparisons of four computer operating systems over a wide variety of loading conditions. The terminal response times, terminal throughput rates, total effective progress rates and batch processing turnaround times are compared under several different conditions. The subjects discussed in this section include the presentation of the CP test results, the TSS test results, the CP and TSS comparisons, the MTS test results, the CP and MTS comparisons, the CP, TSS and MTS comparisons - all under terminal loads -, followed by the batch test results and, finally, the TSS, MTS and OS comparisons under batch loading.

The terminal benchmarking results and comparisons include the terminal response times for each script and the measured and the calculated throughputs and the effective progress rates over a wide range of loading conditions. Sometimes the results are presented as average per terminal, per script or as a total over all scripts. For CP the percentage of problem time and paging rates are presented as secondary performance results.

The batch benchmarking results and comparison include the elapsed time to complete the three benchmarks under various configurations of TSS, MTS and OS.

On the other hand, the preliminary results of the mixed script tests and the terminal probe tests are presented previously in Section 5. The results include a performance comparison with mixed script loading and with fixed script loading as used in the benchmarking tests, and the comparison of the benchmarking loads with the normal terminal loads that exist under actual operating conditions.

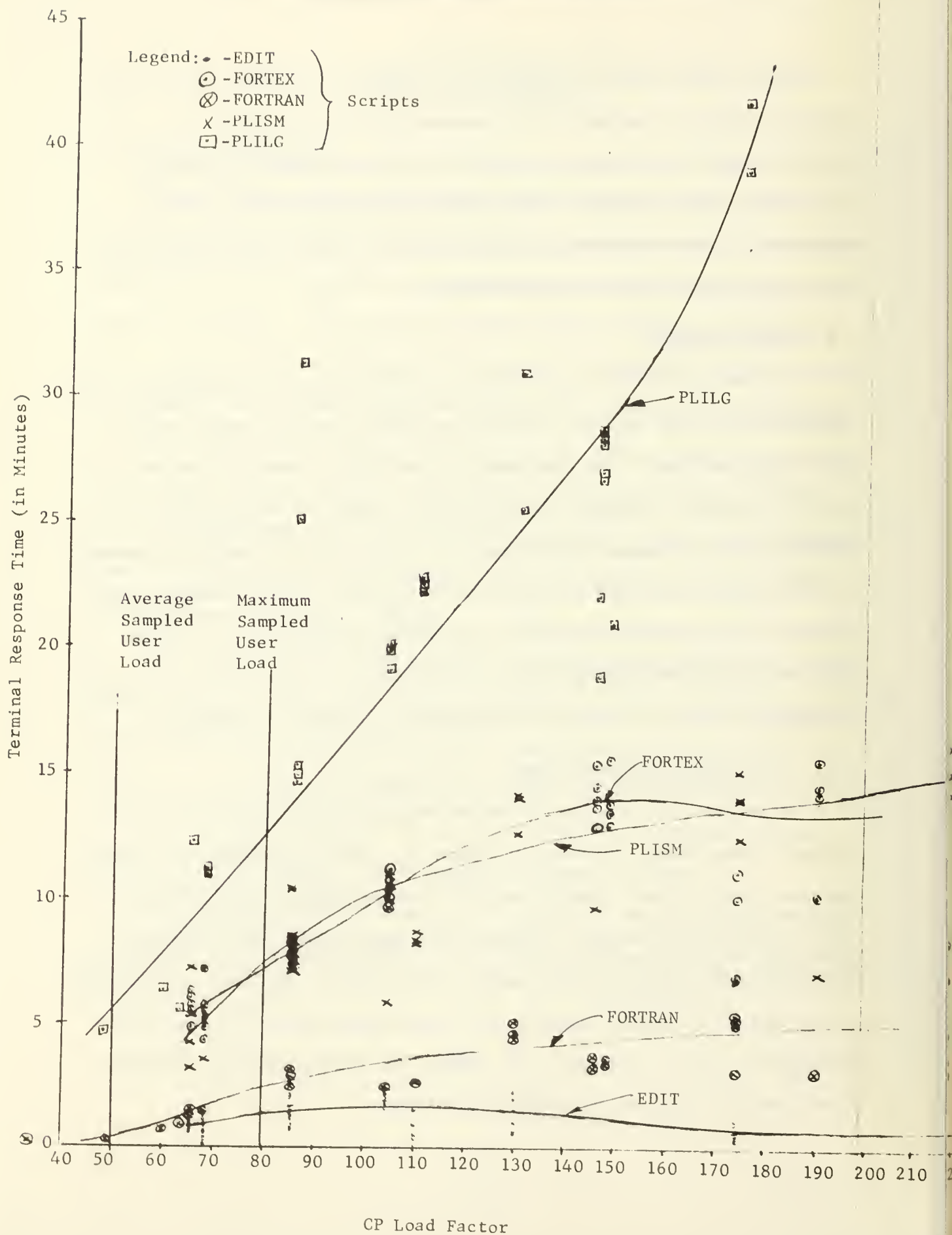
6.2 CP Test Results

This section presents the results of 22 CP terminal benchmarking runs representing a wide variety of loads from fairly typical normal loads to very heavy overloads. The assignment of scripts to terminals for each run are presented previously in Section 3.1, while the description of the scripts and the programs are in Sections 2.2.1 with the listing in Appendix A. The results presented in this section include terminal response times, measured and calculated throughputs, effective progress rates, problem time percentages and paging rates. The results are almost always presented in graphical form; the numerical equivalents are shown in Sppendix C.

6.2.1 CP Terminal Response Times

The CP terminal response times for various CP load factors for all scripts (except PAGE) are shown in Figure 2. The data plotted is the mean response time for each terminal and as shown in Table C1 of Appendix C for each run, the PLILG response increases the most rapidly and is almost a linear function of the CP load factor. Of course the linearity is partially due to the method of defining the load factor but, since the consistency of points is not a function of the chosen load factor, the PLILG response curve serves to support the load factor concept.

Figure 2: CP TERMINAL RESPONSE TIME FOR EACH SCRIPT
UNDER VARIOUS LOADS



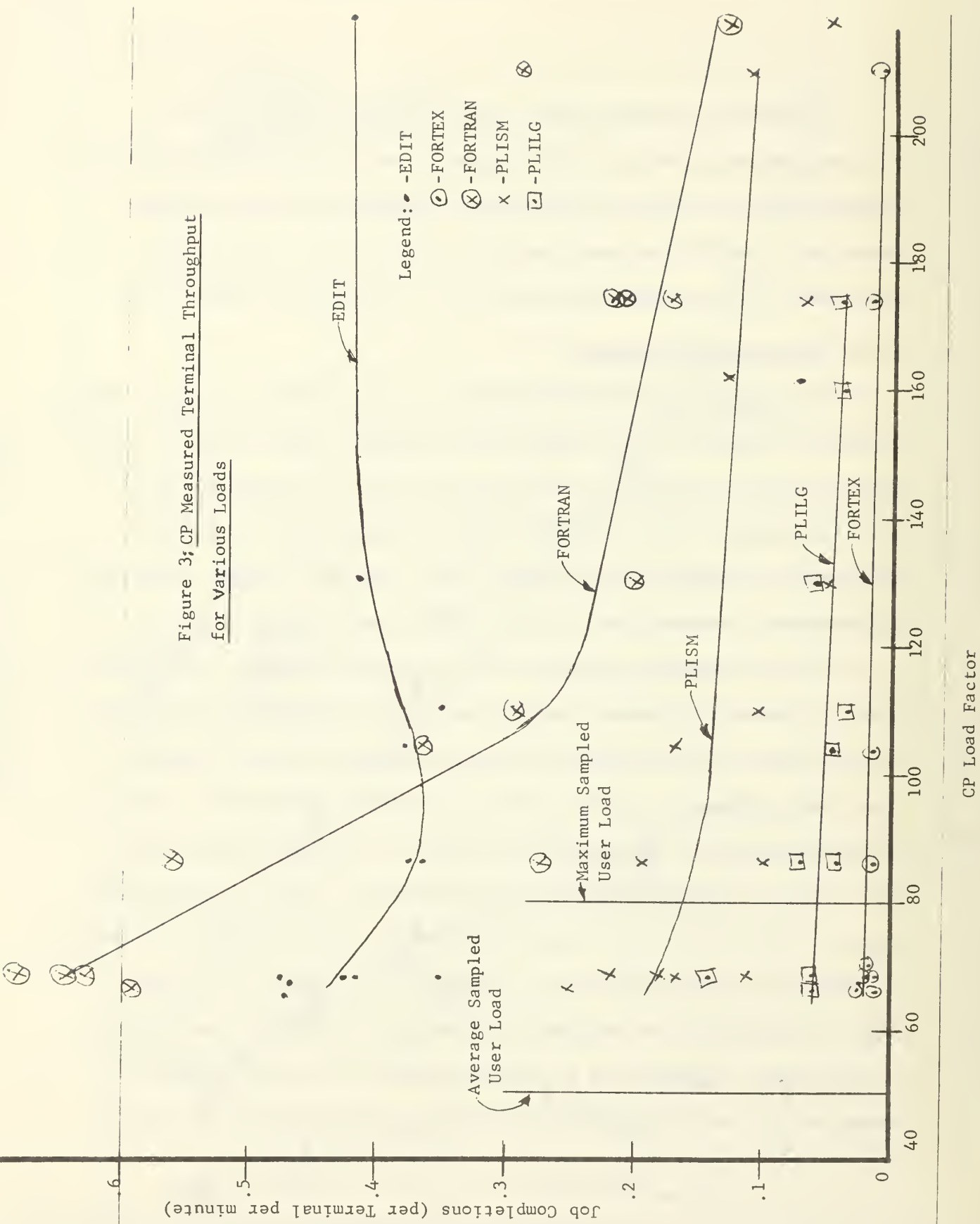
The FORTEX and PLISM response times are approximately the same and they are the second and third highest curves. The FORTRAN response times curve are next and are quite flat, especially for load factors above 140. The EDIT response time is the lowest and is the only one that peaks at an intermediate load, i.e., at load factor equal to 100.

6.2.2 CP Measured Throughputs

The CP measured terminal throughput for each script for various loads is shown in Figure 3. The FORTRAN mean throughput drops the most rapidly from 0.6 jobs per minute at a load factor of 70 to 0.1 jobs per minute at a load factor of 200. FORTEX, PLISM and PLILG all have a gradually decreasing throughput with increasing load. The EDIT throughput remains approximately constant over all loads with a small dip near 100.

The total measured throughput for all terminal running a particular script is shown in Figures 4 and 5. The data presented in these graphs was obtained by multiplying the average throughput for each terminal (as shown in Figure 3) by the number of terminals running that script. The wide variation in throughput per script for any particular load is the result of a shifting of the load percentage in favor of a particular script (and therefore the resources allocated to the script) as the number of terminals running that script is changed. For example, if a large number of terminals are assigned PLILG scripts (i.e., 14 on Run R45 with a load factor of 146), then a large percentage of the total computer resources are assigned to PLILG scripts and the resulting total PLILG

Figure 3; CP Measured Terminal Throughput
for Various Loads



CP Load Factor

2.02
1.65
Figure 4: CP Total Measured Throughput for Each Script

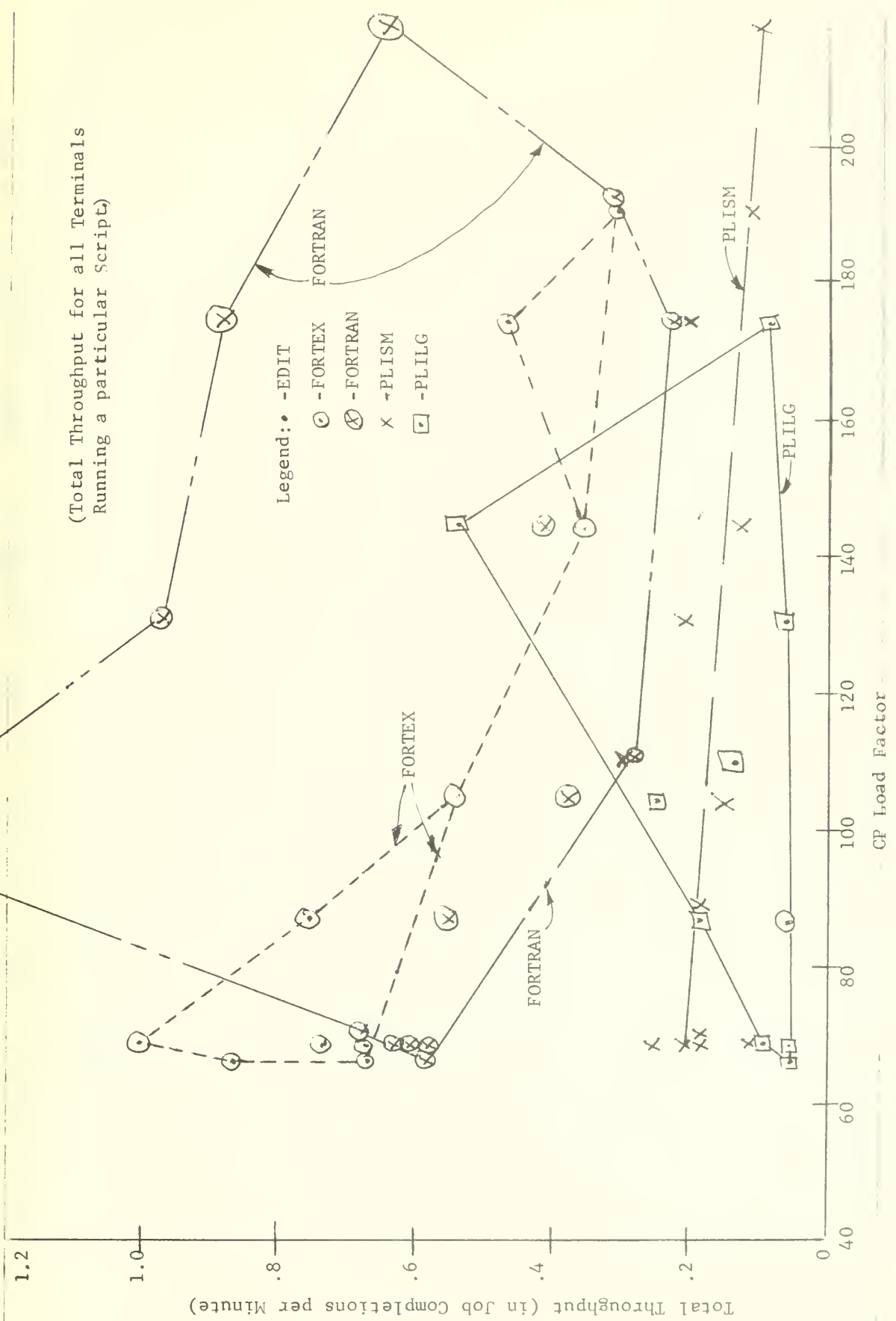
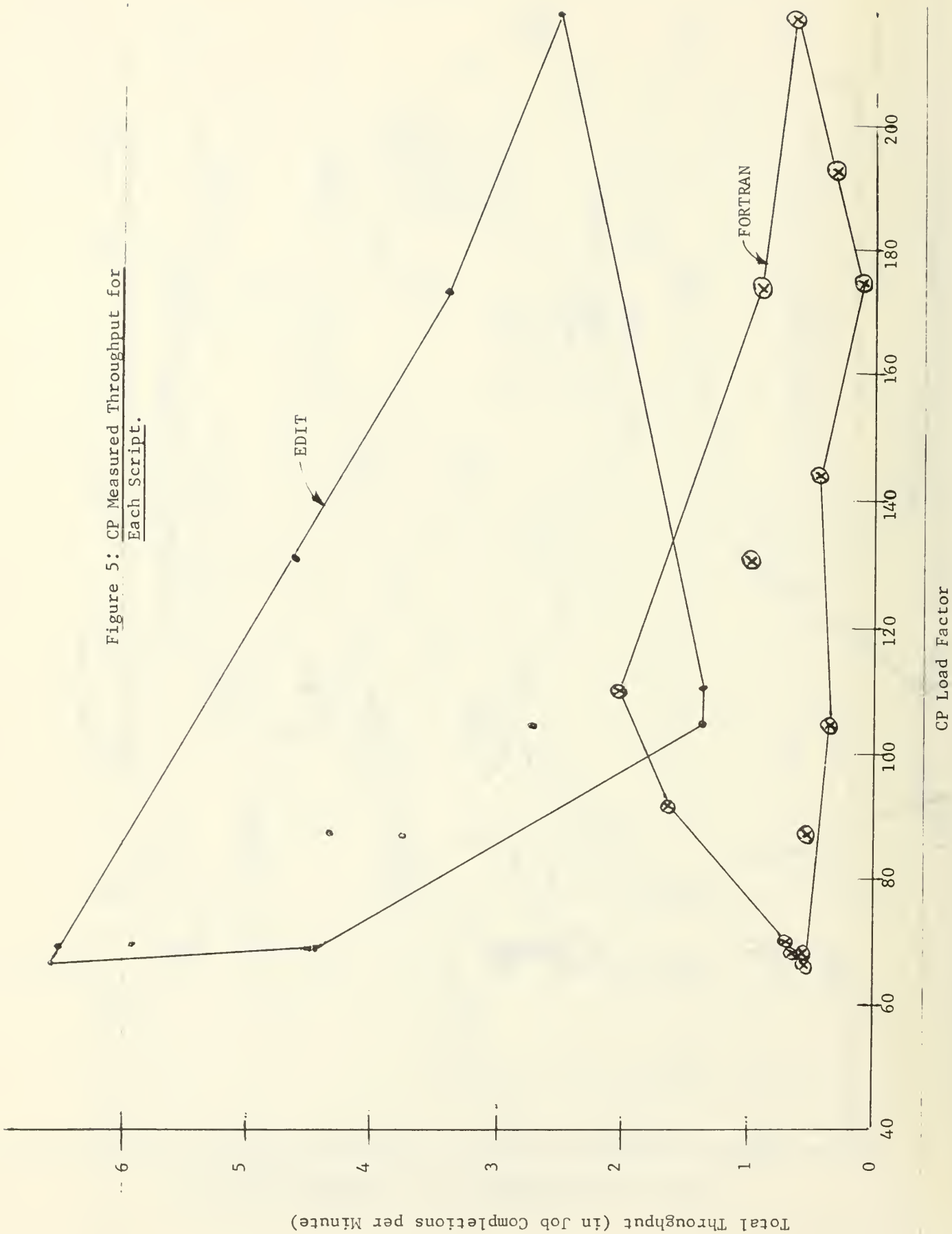


Figure 5: CP Measured Throughput for
Each Script.



throughput is large. Since there is such a wide variation in throughput, the "envelop-type" graphs have been used in several cases on Figures 4 and 5.

The total measured throughput for CP is shown in Figure 6. The total measured throughput was calculated by adding the five total measured throughput per script (as shown in Figures 4 and 5).

The total throughputs for two weighting factors are also shown in Figure 6. For example, the curve for the CP load factor weighted throughput is determined by:

$$TP = \sum_{i=1}^6 \omega_i TP_i$$

where ω_i is the CP load factor weight for script i and TP_i is the total throughput for script i from Figures 4 and 5. Thus a script such as PLILG which has a large load factor weight and low throughput rate has its total throughput multiplied by a large factor. As can be seen from Figure 6 the extra weighting for the heavy loading jobs has little effect on the shape of the total throughput curves.

6.2.3 CP Calculated Throughputs

The CP mean calculated throughputs were determined using Equations 8 and 9 in Section 4. The CP mean calculated terminal throughputs are plotted in Figures 7 and 8 using the dashed lines. The throughputs for EDIT and FORTRAN scripts are shown in Figure 8 on a larger scale. The PLISM and EDIT throughputs varies significantly and are shown using envelop graphs.

Also shown on Figures 7 and 8 are the measured terminal throughput curves (not individual points) in solid lines, so that they may be compared to the

Figure 6. CP Total Measured Throughput with Various Weighings

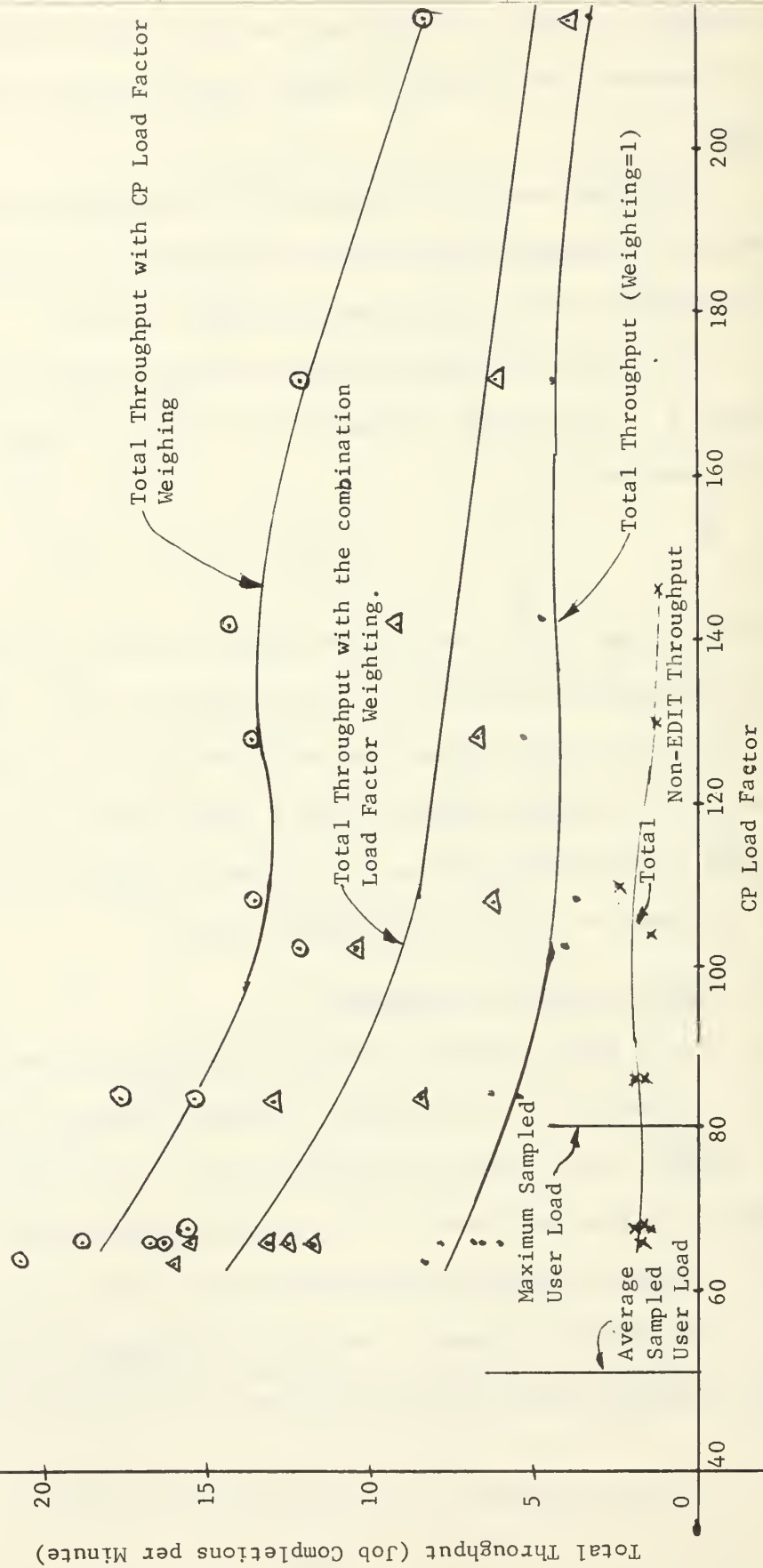


Figure 7: CP MEAN CALCULATED THROUGHPUT vs MEASURED TERMINAL THROUGHPUT-
FORTEX, PLISM, PLILG

Legend: ○ -FORTEX

x -PLISM

□ -PLILG

--- DASHED LINE-CALCULATED

— SOLID LINE-MEASURED

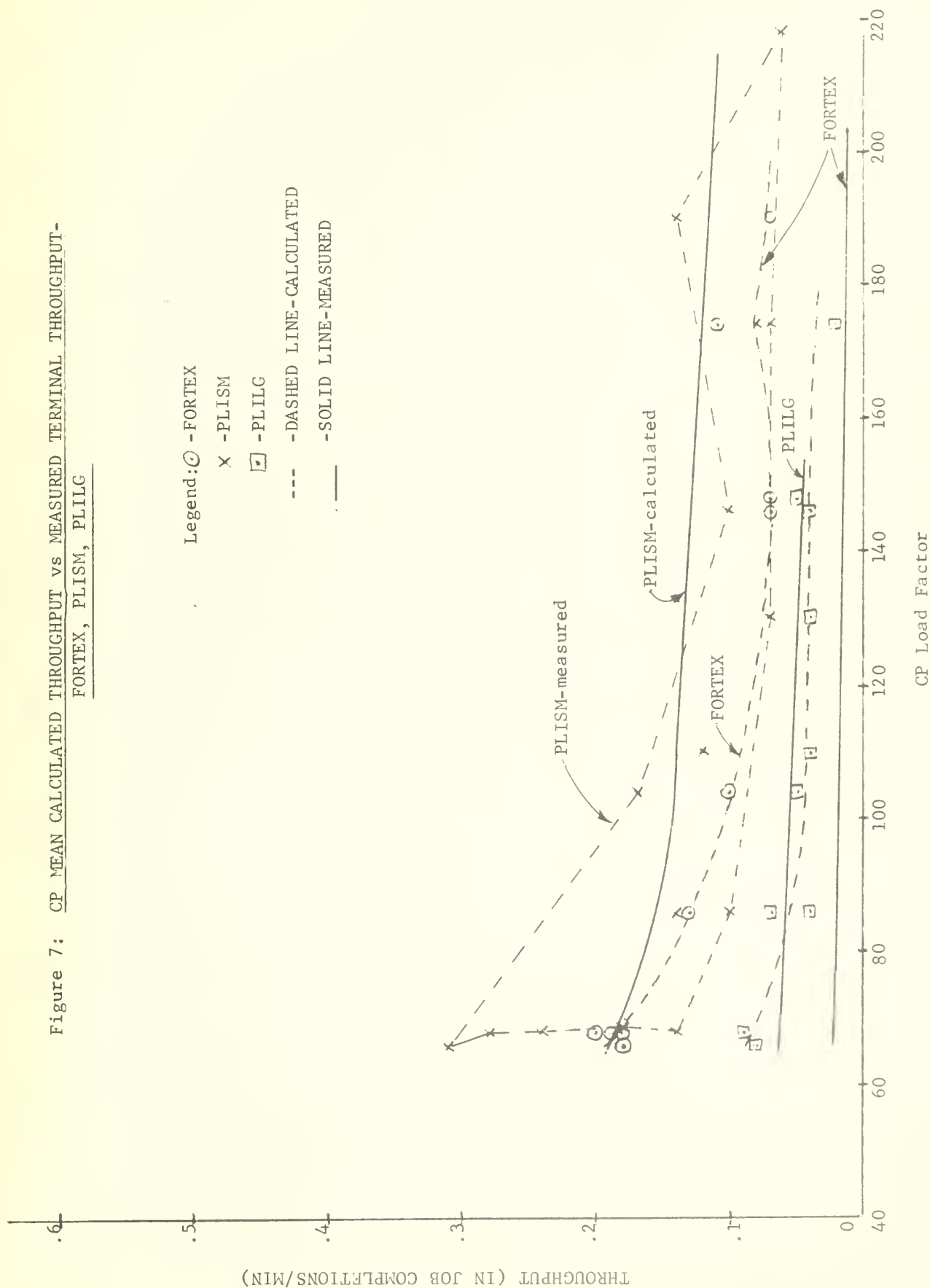
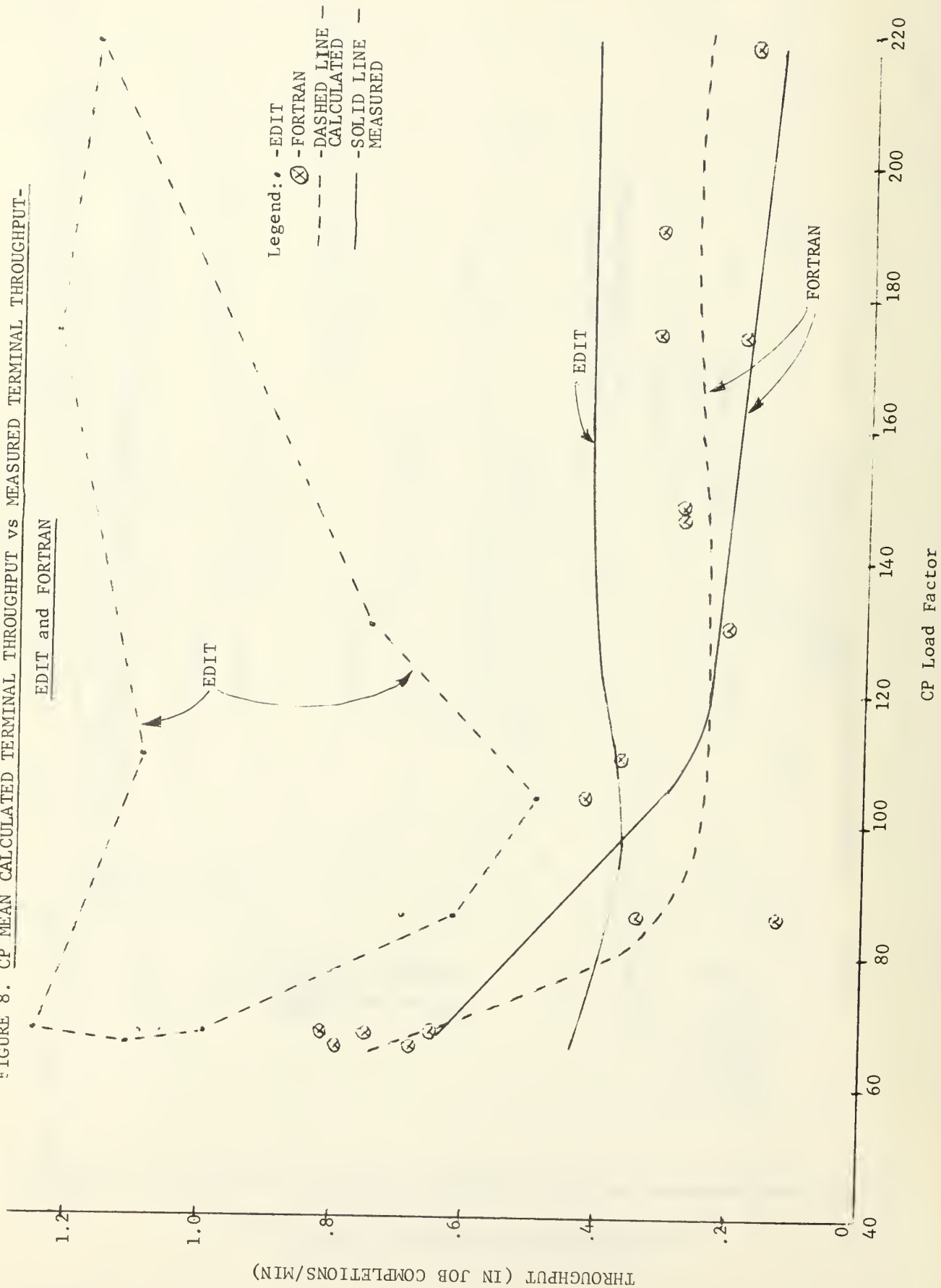


FIGURE 8. CP MEAN CALCULATED TERMINAL THROUGHPUT vs MEASURED TERMINAL THROUGHPUT-



calculated throughput. As expected the calculated throughput is generally higher than the measured throughput. This is particularly true for short-execution programs, such as EDIT and FORTEX, where a large part of the total elapsed time is involved in the overhead of running the scripts and calling the clock rather than executing the actual programs. The reasons for the calculated throughput sometimes being lower than the measured throughput is due to some errors in measurement and the fact that mean values are being plotted whereas there is often large variance in the actual data points. The actual data points from which these graphs were made are in Appendix C in Table C3.

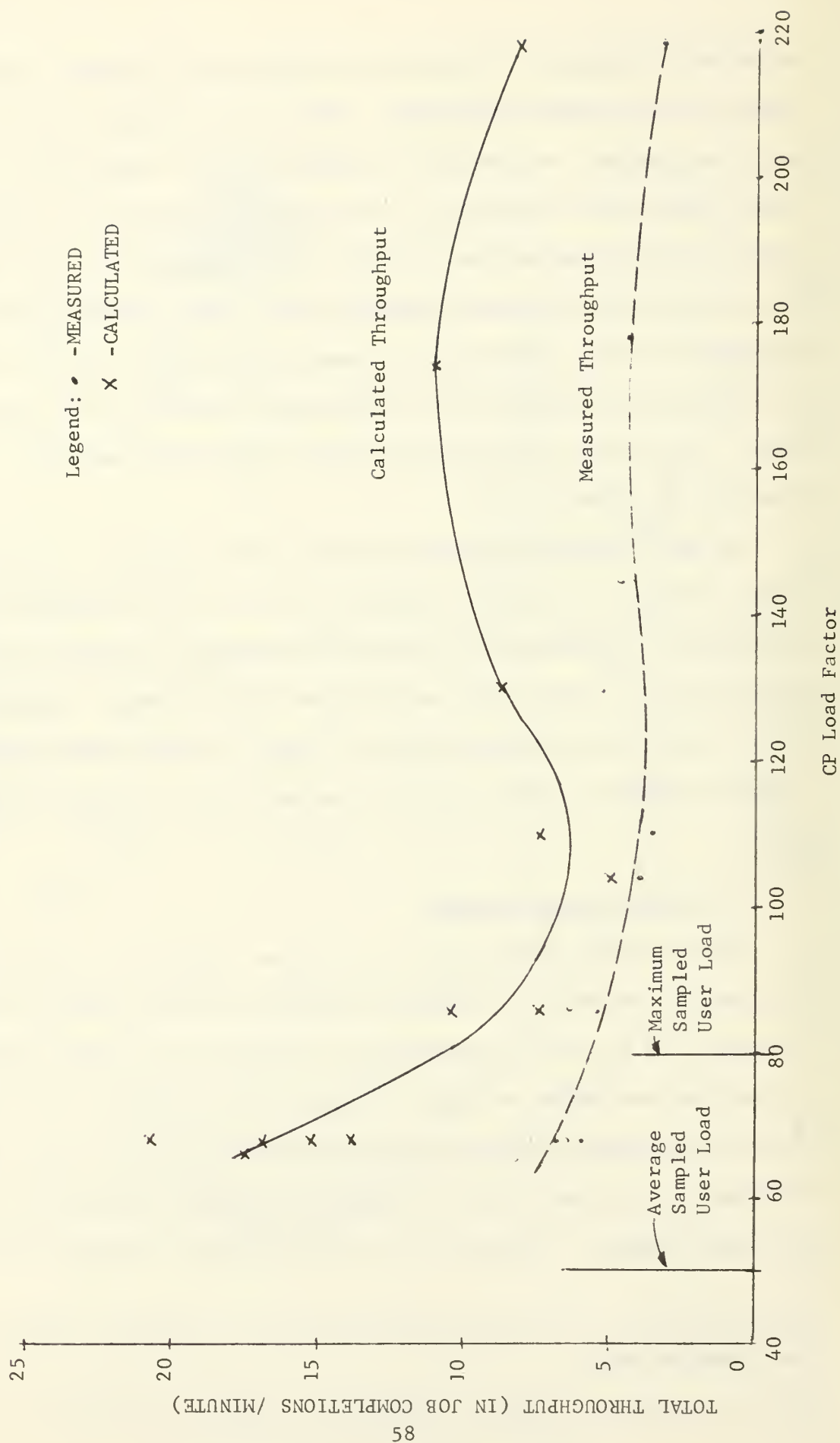
The total measured throughput and the total calculated throughput are shown in Figure 9. The points represent the sum of the throughputs for all the scripts. As expected the calculated throughput is larger than the measured values. Both curves have the same general shape - dropping quickly as the load factor increases from 68 to about 110, then rising slightly and finally falling off again as the load increases into the heavy overload region.

6.2.4 Effective Progress Rates

The effective progress rate is on another indicator of the performance of a computer system. The effective progress rate was originally defined by Lasser [15] as the time required for a program to run stand alone divided by the time required to run under the given load condition.* If the sum of effective progress rates for all programs of a particular type currently being executed is one, then the computer is operating as well as a serial processing computer. The amount over one indicates the degree to which the

*The reciprocal of the effective progress rate is sometimes referred to as the dialation factor.

Figure 9: CP TOTAL MEASURED AND CALCULATED THROUGHPUT



performance is improved by overlapping the use of various resources under multiprogramming. Effective progress rates less than one indicate the system is operating more poorly than a serial processing computer due to heavy overloading.

The effective progress rate for each script is presented in Figure 10. The most obvious observation from Figure 10 is that all the points for all scripts are grouped together. The effective progress rates for the scripts drop rapidly as the load factor increases from 40 to 90 but then remain relatively constant at between 0.02 and 0.05 as the load increases above 90.

The total effective progress rate, as presented in Figure 11, is almost constant at 0.8 for load factor from 85 to 150. This means the computer is operating at 80 percent of the throughput of a serial, one-job-at-a-time, processor. For load factors less than 85, the total effective progress rate rises rapidly to about 1.2 at 65. Unfortunately, there is not sufficient data to determine the total effective progress rate at lower load factors where the multiprogramming advantages are more noticeable.

In an attempt to predict the effective performance at lower loading conditions, some speculation was done on the probable terminal response time at the low loads. Using Figure 2, it is possible to speculate on the terminal response times for light loads such as runs R21 to R25. Assuming (1) that the response times are linear from the lower end of the curve to the zero load factor point, (2) that the response times are at their minimum (or stand-alone) values as given in Table C6 in Appendix C at zero load

Figure 10. CP Effective Progress Rates
per Script

Legend: • EDIT
 ⊙ FORTEX
 ⊗ FORTRAN
 × PLISM
 □ PLILG

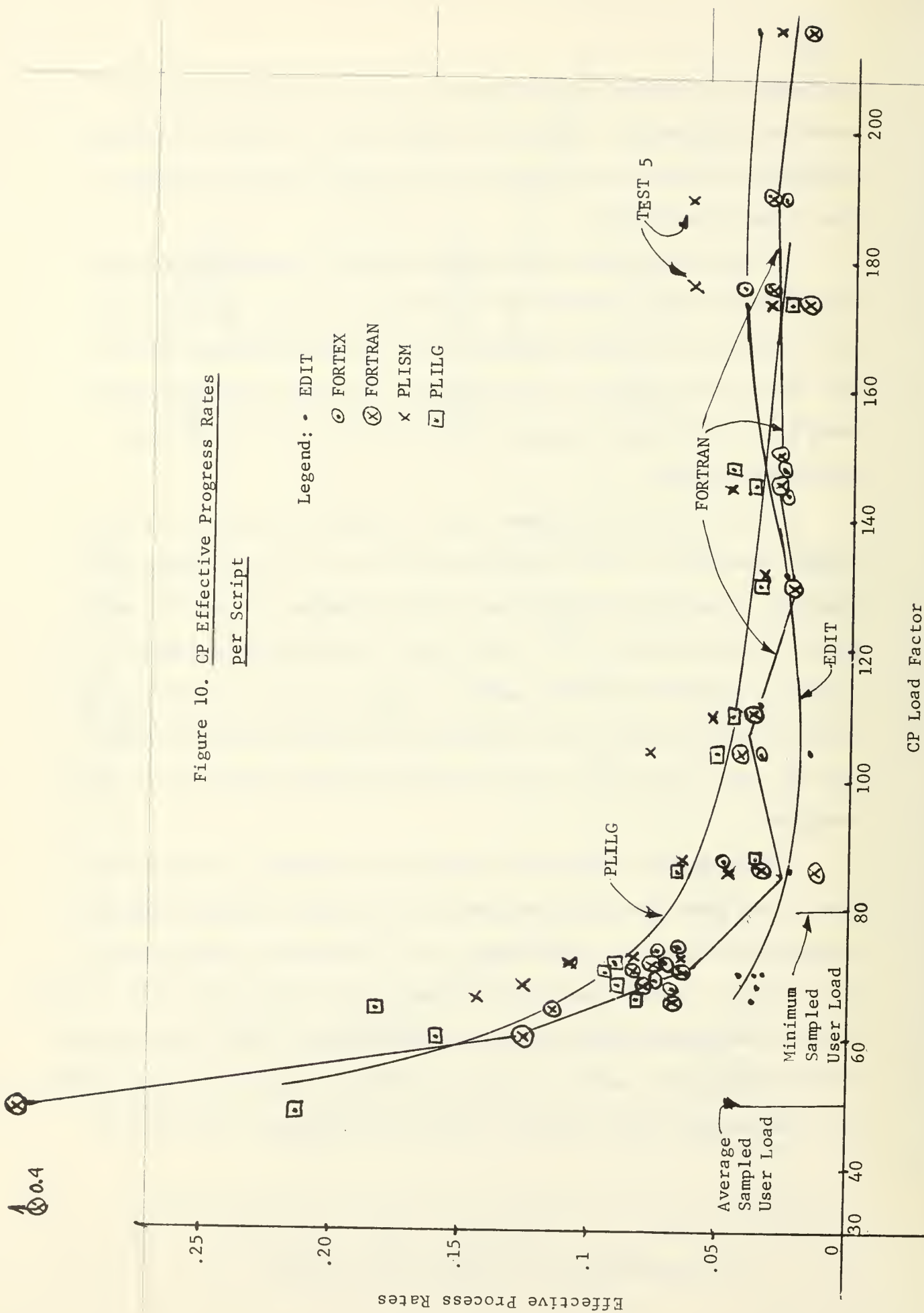
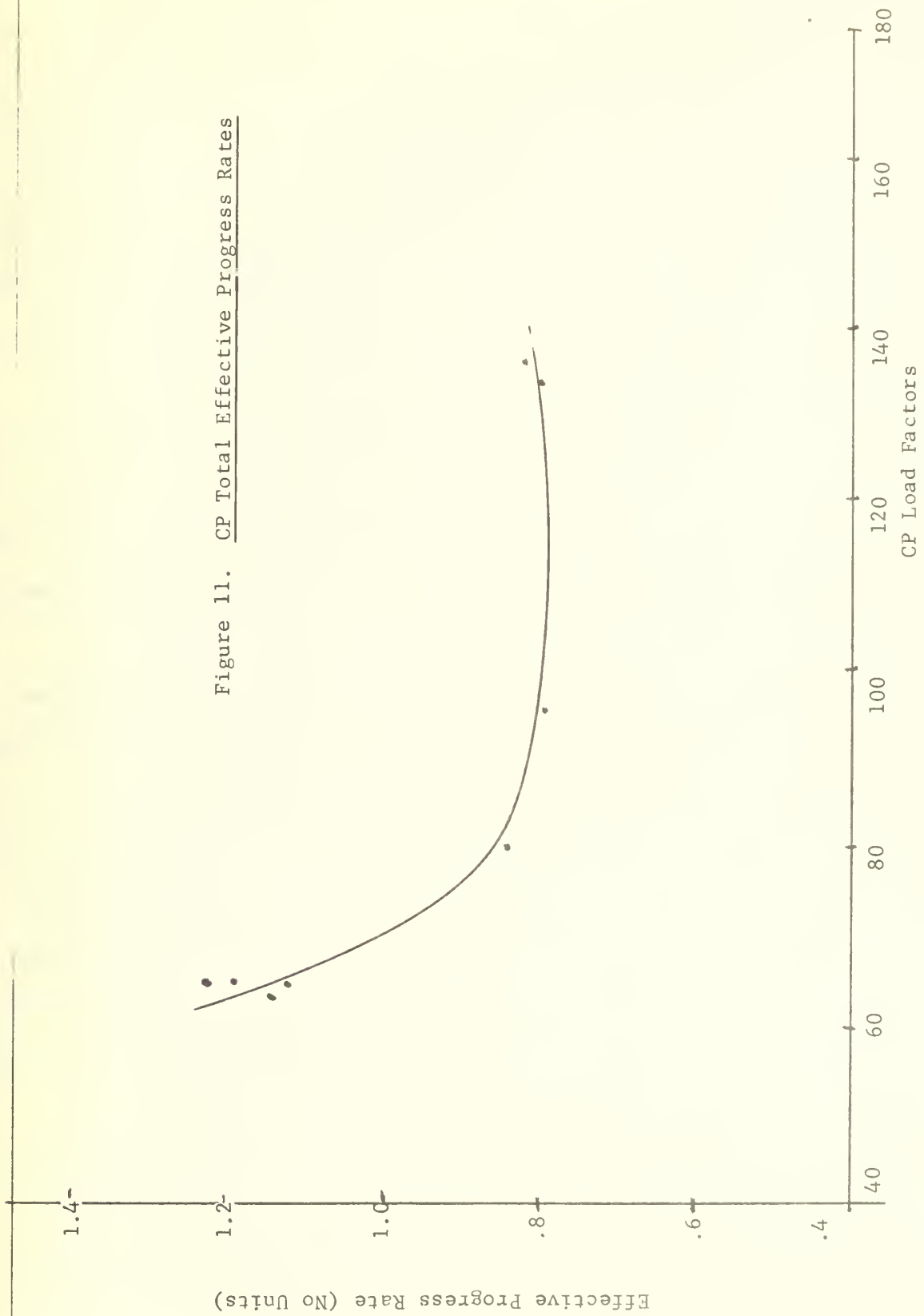


Figure 11. CP Total Effective Progress Rates



factor point, and (3) that the EDIT curve stays below the FORTRAN curve in Figure 2, then it is possible to predict (approximately) the terminal response times for EDIT, FORTEX and PLISM scripts at low load factors. The conjectured response times for runs R21 to R25 are shown in Table 9.

The effective progress rates were then calculated from the conjectured response times and are also shown in Table 9. The conjectured effective progress rates range from 1.5 at a load factor of 64 to 3.8 at a load factor of 32. Unfortunately, these values are very dependent on the EDIT response times - since there are 8 to 15 EDIT scripts running. Therefore the EDIT response times may change the effective progress rates significantly especially for low load factor where it is difficult to predict the edit response times accurately.

Since the normal operating load factor was determined to average about 50 with peak to 80 (Section 5.4), Table 9 suggests that the CP system normally operates with a total effective progress rate of 3, but Figure 11 shows that rate may drop to as low as 0.8. In other words, during normal operation it is conjectured that the CP time sharing system processes jobs about three times as fast as it would if it operated sequentially (no multiprogramming and no overlapping of input/output and processing). However, during peak loads the CP system processes jobs at only 0.8 times as fast as it would if it processed jobs sequentially.

It seems from the above evidence that it behoves the designer of CP system to ensure that system does not accept new jobs when it is approaching, or already over, the critical load factor. This subject will be discussed further in a later section.

Table 9: Total Conjectured Effective Progress Rates - CP

Item	Run	R21	R22	R23	R24	R25
CP Load Factor		32	32	48	60	64
EDIT response		.1	.1	.2	.5	.7
FORTEX response		1.5	1.5	2.5	3.8	4.2
PLISM response		1.5	1.5	2.5	3.8	4.2
EDIT Progress Rate		.3*	.3*	.17	.07	.05
FORTEX Progress Rate		.25	.25	.15	.1	.09
PLISM Progress Rate		.3	.3	.2	.12	.11
EDIT Progress rate times Number of EDITS		2.4*	3.0*	2.0	.98	.75
Total Conjectured Progress Rates (Also uses Table C7)		3.5*	3.8*	3.0	1.8	1.5

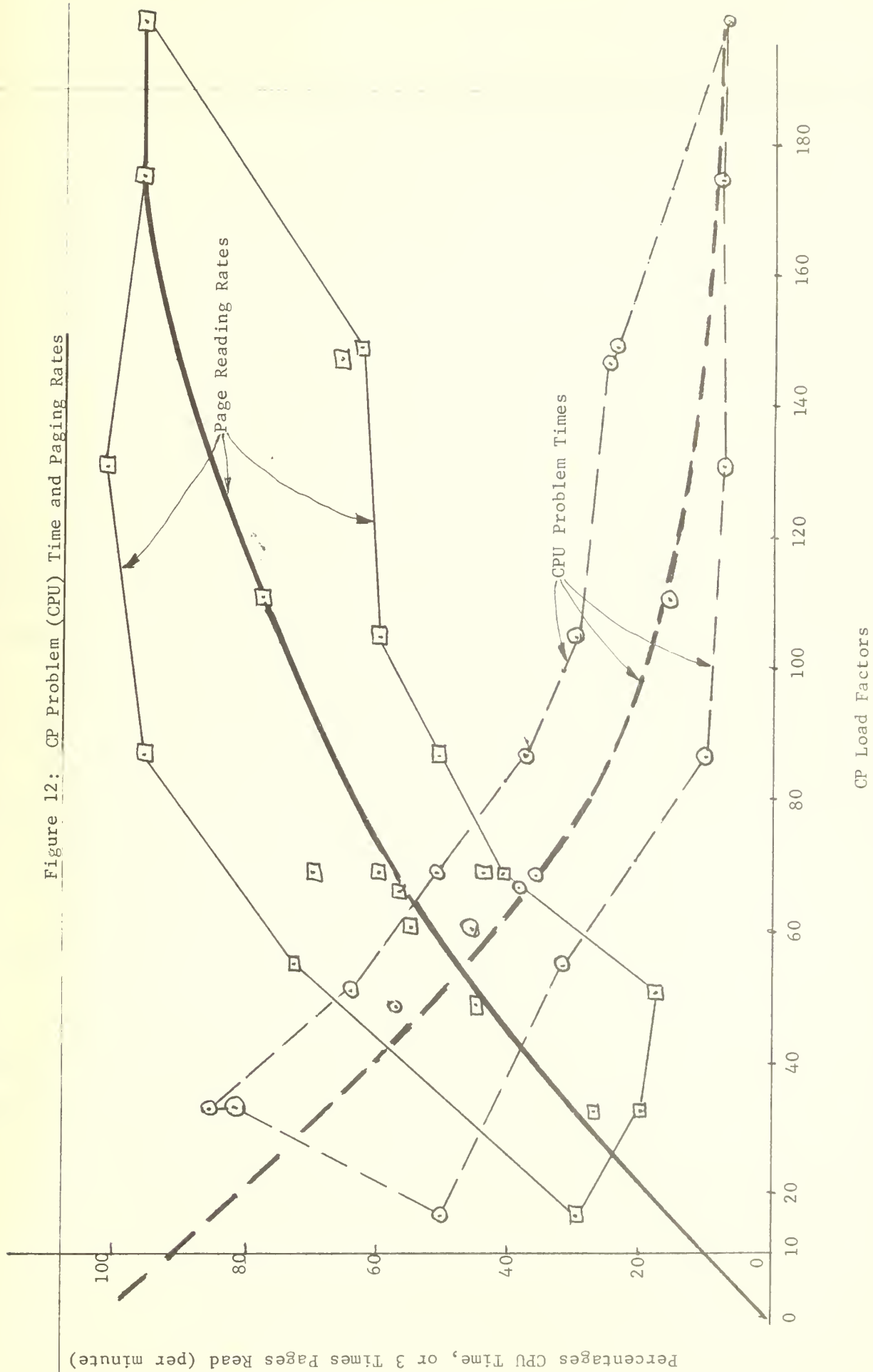
*These EDIT effective progress rate may be in error and may be as low as 0.2 or as high as 0.7. For the 0.2 value, the total conjectured effective progress rate would then be 2.7 and 2.8 for runs R21 and R22, respectively; for the 0.7 value, the total progress rate would be 6.7 and 7.8 for runs R21 and R22 respectively.

6.2.5 CP Software Monitor Results

The secondary performance measurements for determining system efficiency and performance were obtained from a CP software monitor called "MEASURE". MEASURE was obtained from Mr. Stuart Wecker of the State University of New York, Stony Brook, New York. MEASURE provided information such as: CPU problem time, supervisor time, overhead time, pages reading rate, pages swapped rate (written on back-up storage), and pages stolen rate (removed from core while the user was in one of the active queues). In addition to the system totals, each user was also monitored for the same information. MEASURE provided information at different intervals and was run continuously while the benchmark programs were under testing. From these observations the percentage of CPU time spent in problem state and the paging rates were used as indications of the load on the system.

The results of a CP software monitor are presented in Figure 12, which shows the percentage of CPU time spent in the problem state and the rate at which memory pages are read from disk storage into fast memory. There is a wide variation in both the CPU problem time and the page reading rate as a function of load factor as shown by the large envelopes in Figure 12. However there is a definite trend as shown by the heavier center curves. The CPU problem percentage starts near 80 percent for a load factor of 20, drops below 50 percent for a load factor of 55 (the average user load), drops to 30 percent for a load factor of 80 (the normal maximum load) and then drops continually to 7 percent. On the other hand the pages-read rate starts at zero and increases to 35 pages per minute. The reason for the

Figure 12: CP Problem (CPU) Time and Paging Rates



rapidly decreasing CPU problem time is the heavy demand for new pages and the limited input/output channel capacity.

The relationship between the problem time and the paging rate is shown more gramatically in Figure 13 which shows that, over all the CP tests conducted, there is a consistent relationship that:

$$\frac{\text{percentage}}{\text{problem time}} = 100 - 3 \times \text{Paging Rate.}$$

This formula shows that to get high CPU utilization (and therefore high throughput), it is necessary to limit the paging rate rather severely.

The percentage of time that the processor spends running the operating system, commonly called overhead, is also of interest. Figure 14 shows the percentage of time that the processor spent in problem state, in problem state plus supervisor state, and in supervisor state for Test 4. Since the load increases from R42 to R46 and then abruptly decreases, the problem time percentage decreases from about 40 percent in Run R42 to about 20 percent in Run R46 due to overloading. However the percentage of supervisor time stayed approximately constant at 13 to 18 percent. The points plotted in Figure 14 are actually successive reading from the software monitor, MEASURE, rather than averages for the runs.

Figure 15 demonstrates the shift of resource allocation as the number of PLILG is increased to effectively increase the total load. The percentage of processor time allocated to FORTEX decreases as the load increases, although the number of FORTEX script was almost constant.

Figure 13: CP PROBLEM TIME vs PAGING RATES

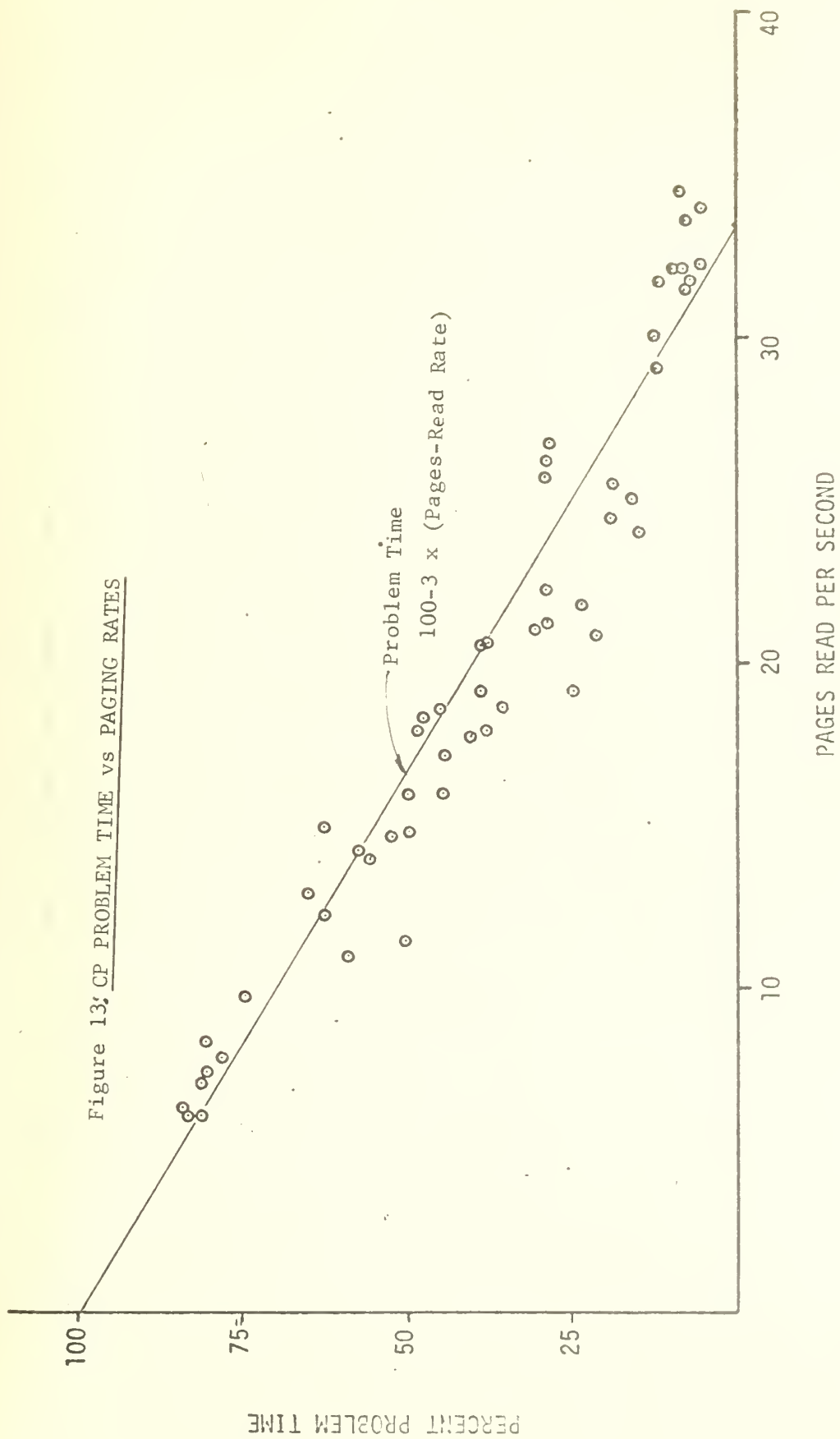
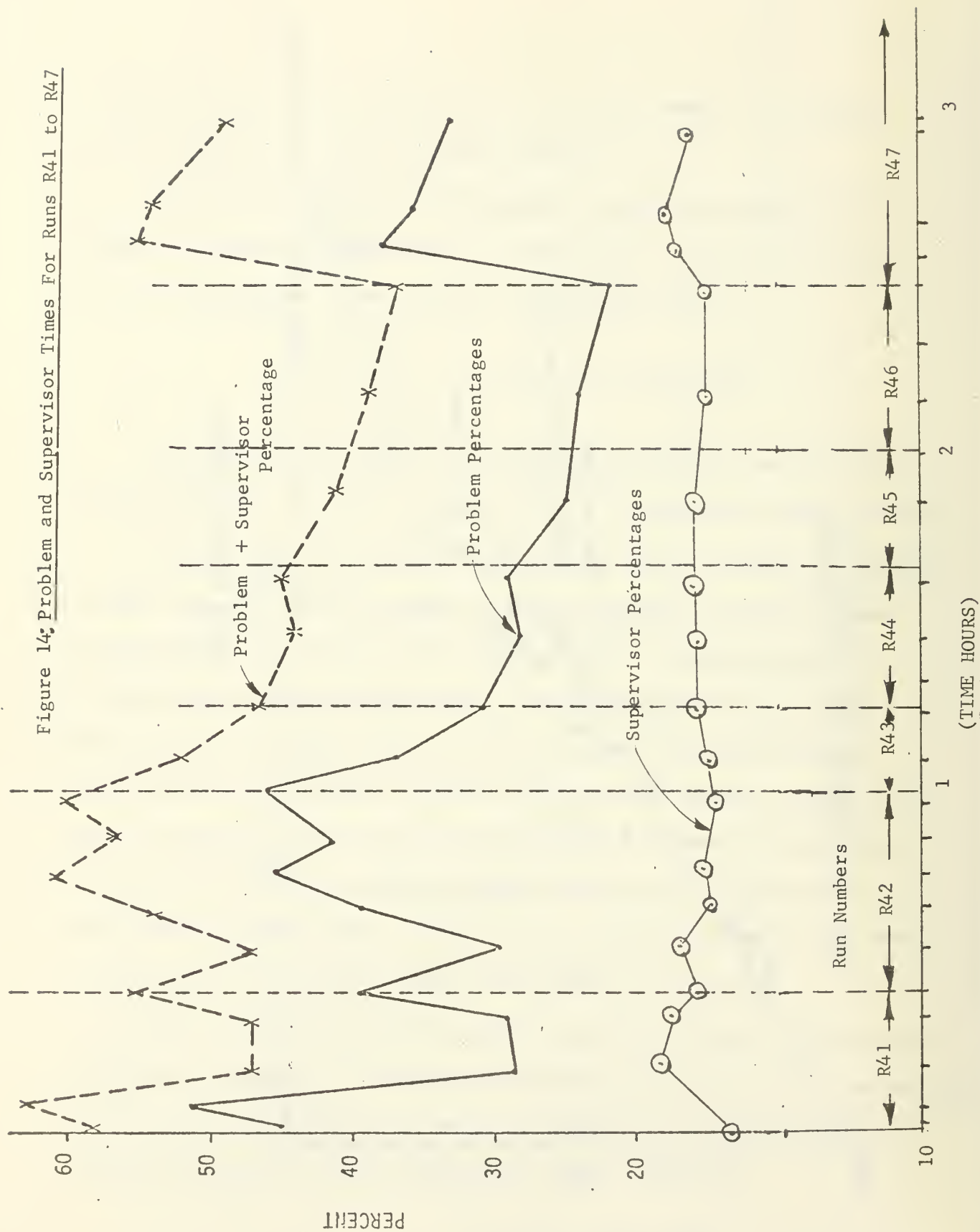


Figure 14: Problem and Supervisor Times For Runs R41 to R47



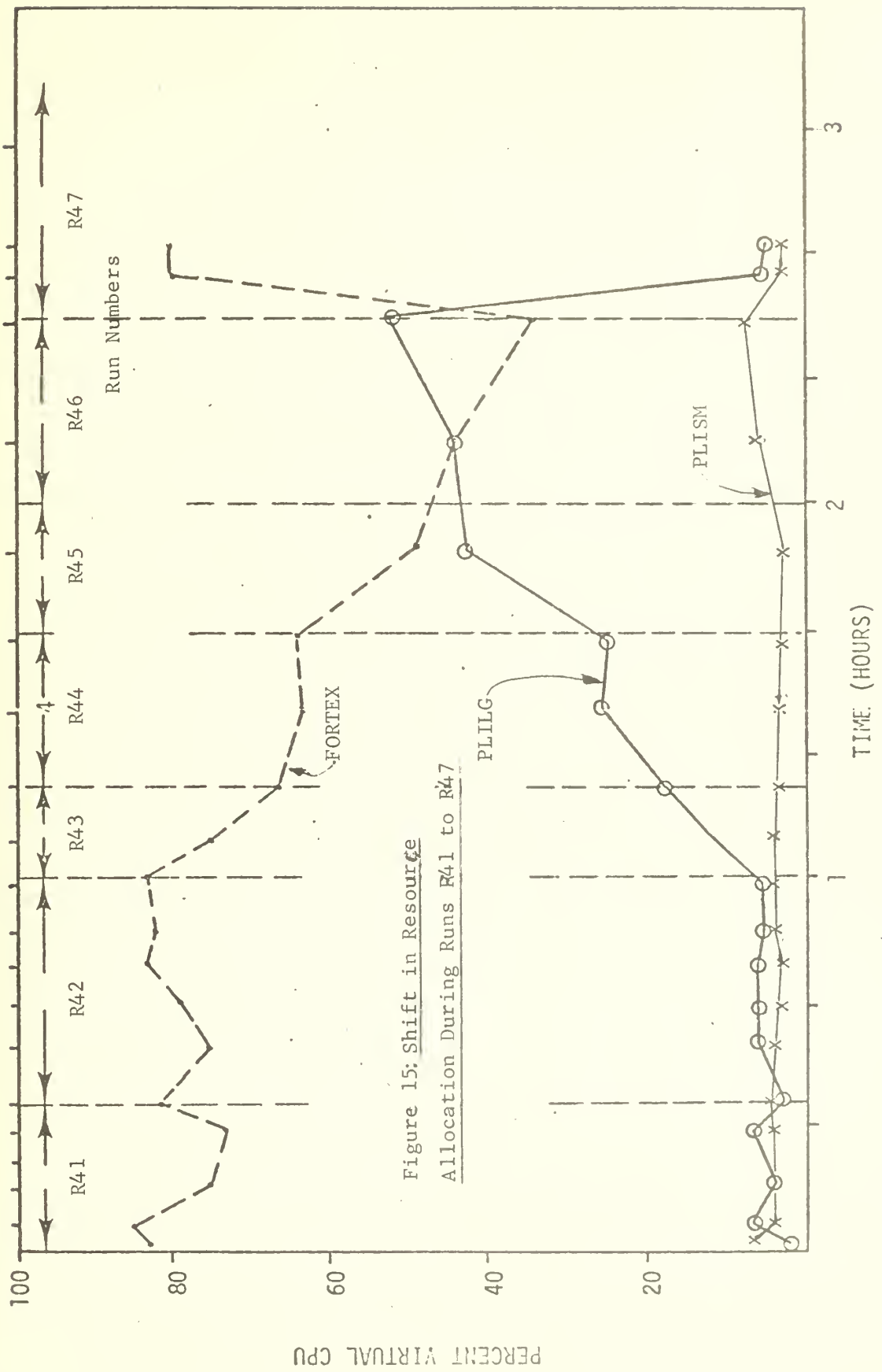


Figure 15: Shift in Resource Allocation During Runs R41 to R47

6.3 TSS Test Results

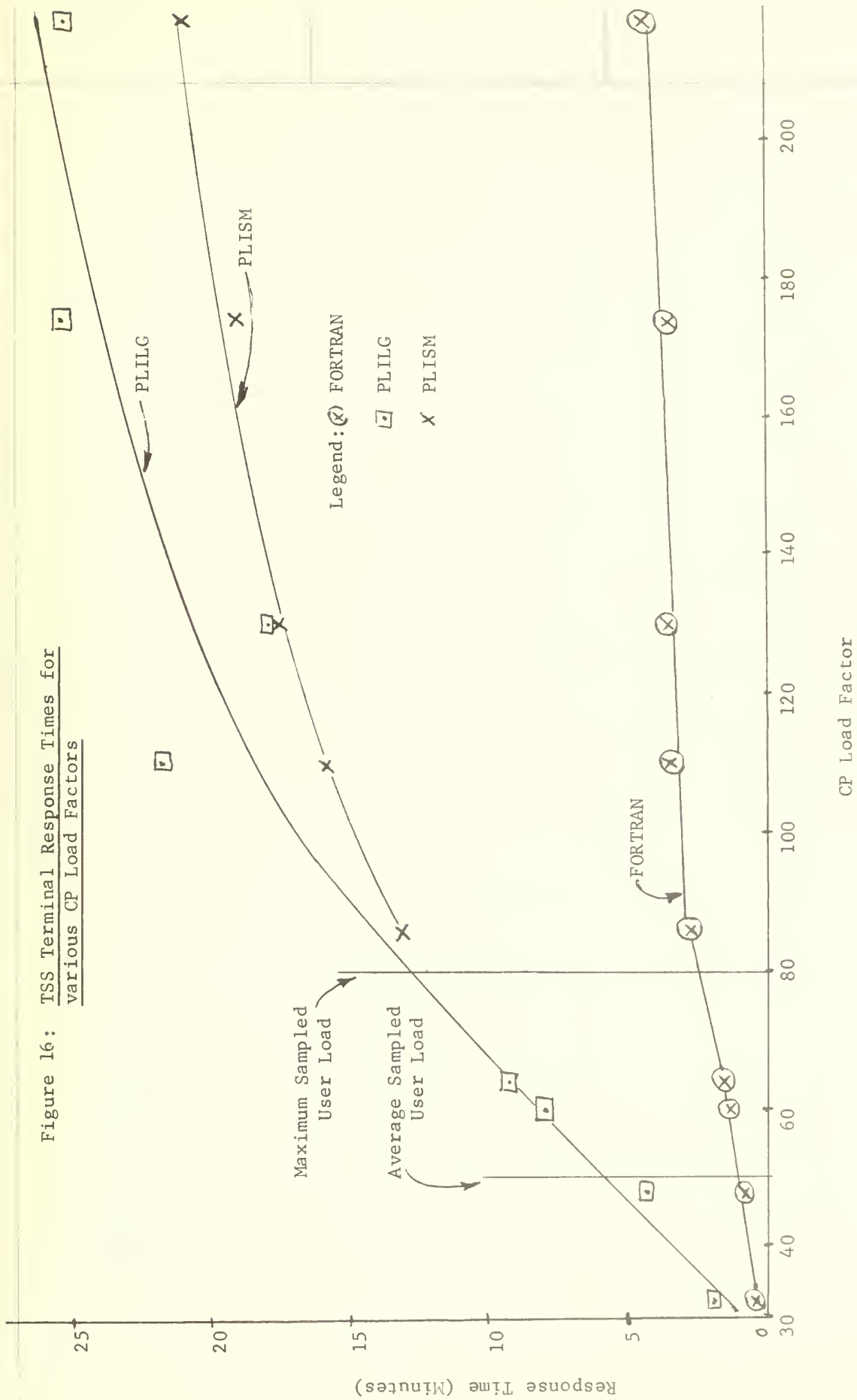
This section presents the results of 9 TSS terminal benchmarking runs under normal to heavy loading conditions. The description of the programs and scripts and the assignment of scripts to terminals during the benchmark runs were presented previously in Sections 2.2.1 and 3.1, respectively. The results in this section include the terminal response times, the measured terminal throughputs and the calculated terminal throughputs. The results are shown in graphical form with the tabular the equivalent shown in Appendix D.

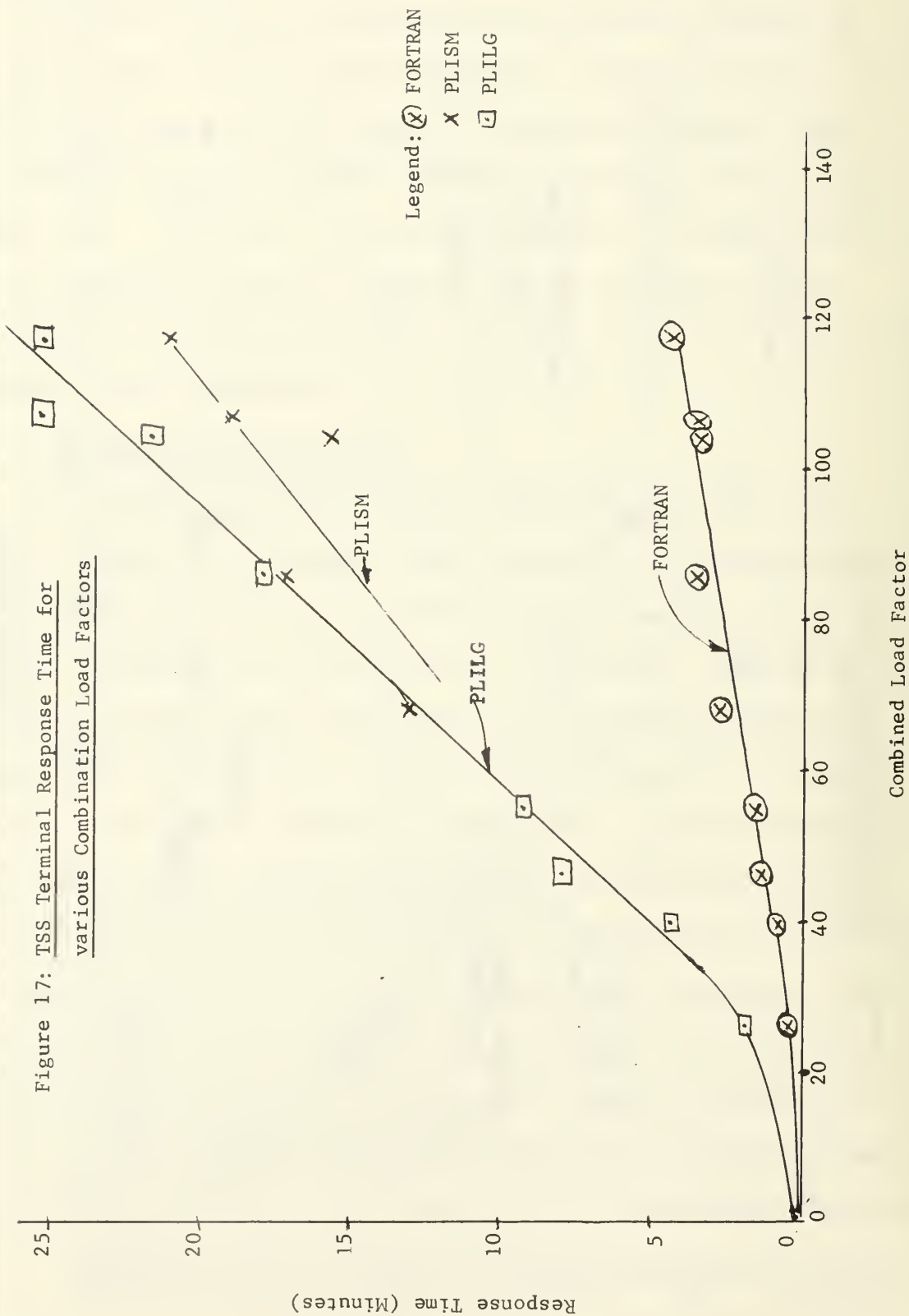
6.3.1 TSS Terminal Response Times

The TSS terminal response times for various CP load factors are shown in Figure 16. The response time curves are fairly smooth functions of the CP load factor but with a definite concave-downward appearance. The PLILG curve shows the greatest degree of curvature. Since the CP load factor appears to have weights that are too large for the TSS response times, these response times were replotted against the combined load factor as shown in Figure 17. In this case, the concave-downward curves are replaced with linear curves again. This demonstrates that the combined load factor is more realistic for TSS than the CP load factor.

The PLILG response as shown is 6 minutes for the average user load, 14 minutes for the heaviest sampled user load and 25 minutes for the heaviest benchmarking load. The corresponding figures for FORTRAN are 1 minute, 2 minutes and 4-1/2 minutes, respectively. The response times for PLISM are only slightly less than for PLILG. Unfortunately the results

Figure 16: TSS Terminal Response Times for various CP Load Factors





from the EDIT and FORTEX scripts were not available from the TSS runs.

6.3.2 TSS Throughputs

The TSS measured and calculated terminal throughputs for the FORTRAN and PLILG scripts are shown in Figure 18. For very light loads (load factor = 30), the TSS throughputs for FORTRAN scripts was 3 per minute, but this quickly dropped to less than 0.5 per minute as the load approached 60. The PLILG response times started at 0.5 per minute and dropped to 0.1 per minute at the load factor of 60. There is very little difference between the measured and the calculated throughputs.

The total measured and calculated throughputs per script are shown in Figure 19. The plotted points were obtained by adding the throughputs for all terminals running the particular script. The points show the same general trend as in Figure 18, except the total is higher and the dropping is less rapid. The FORTRAN points are deceptively close to the curves, because most of the runs were made with the same number of FORTRAN scripts.

The total measured and calculated throughput is not shown because no data is available for the EDIT and FORTEX scripts. A portion of the total throughput will be presented later in Figure 24 as a comparison with CP. Also the effective progress rates are not shown because the data on the TSS stand-alone response time was not available.

6.4 CP and TSS Performance Comparisons

6.4.1 Terminal Response Time Comparisons

The performance comparison of CP and TSS over a wide range of terminal

Figure 18: TSS Terminal Throughputs-
Measured and Calculated

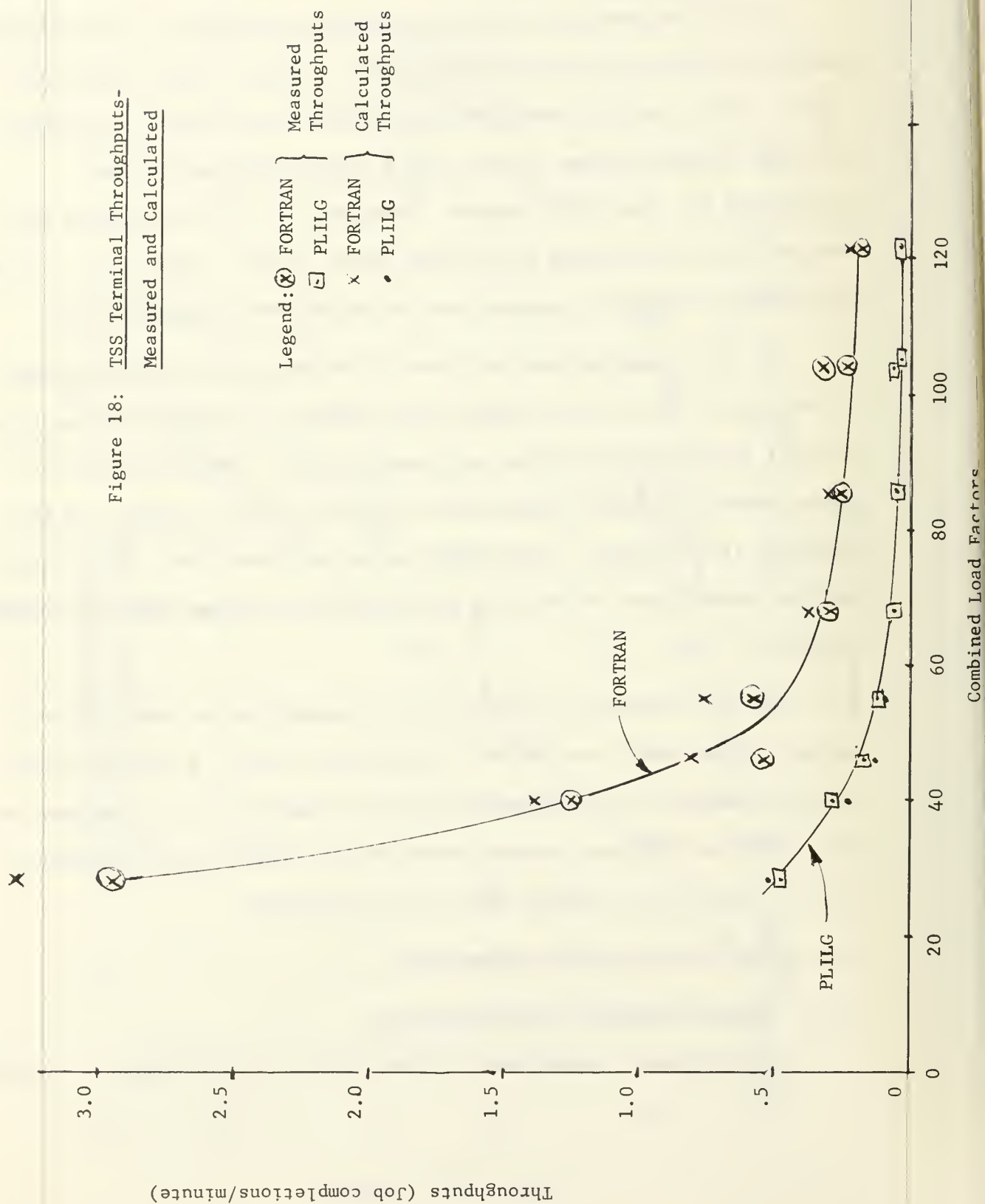
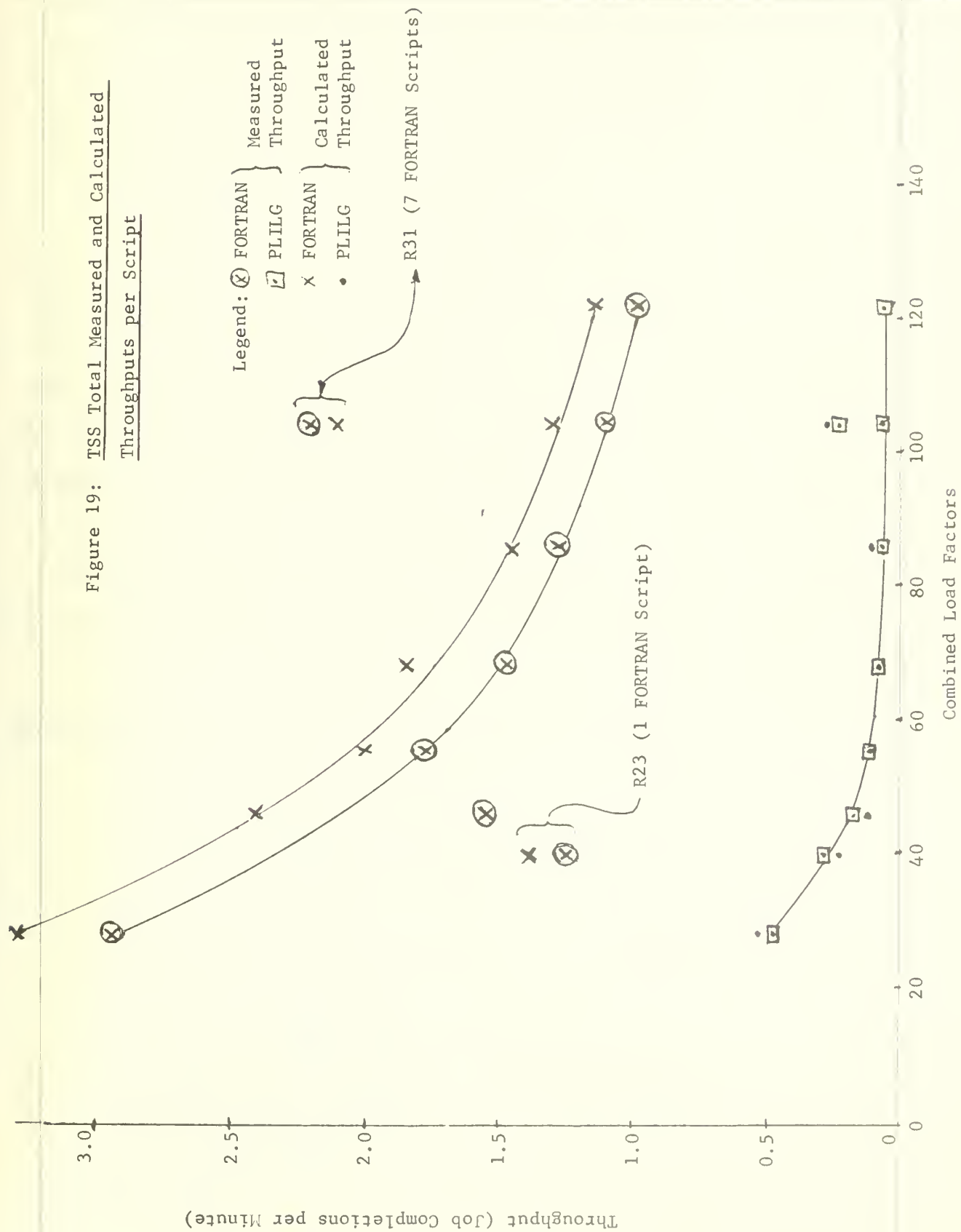


Figure 19: TSS Total Measured and Calculated Throughputs per Script



loading conditions was of primary importance in this project. Since it was necessary to use a common load indicator in order to compare CP and TSS, the CP terminal response times were replotted against the combined load factor as shown in Figure 20. When Figure 20 is compared to Figure 2, which is the same data except plotted against the CP load factor, it is obvious that there is a much wider spread in the response times in Figure 20, especially for PLILG at high loads.

A comparison of the CP and TSS response times is shown in Figure 21, which is a combination of the CP response curves from Figure 20 and the TSS response curves from Figure 17. The PLILG and FORTRAN response times are about the same for CP and TSS, except that the CP PLILG responses have a lot more variation than the TSS equivalents at high loads. For example, at a combined load of 120 the CP response times may vary anywhere from 18 minutes to over 40 minutes, compared to a relatively predictable 26 minutes for TSS. On the other hand, the PLISM responses for CP are all much less than those for TSS.

HOWEVER, the comparison above is for two very different configurations as shown in Table 10.

Table 10: CP versus TSS Comparison Configurations

TSS	CP
Dual processor	Single processor
768K bytes memory	256K bytes memory
3 core boxes	1 core box
1 Drum	1 Drum
2314 Disk (fast)	2311 Disk (slow)

Figure 20 CP Terminal Response
Times versus Combined
Load Factors

Terminal Response (minutes)

Legend:

- -EDIT
- -FORTEX
- ⊗ -FORTRAN
- x -PLISM
- -PLILG

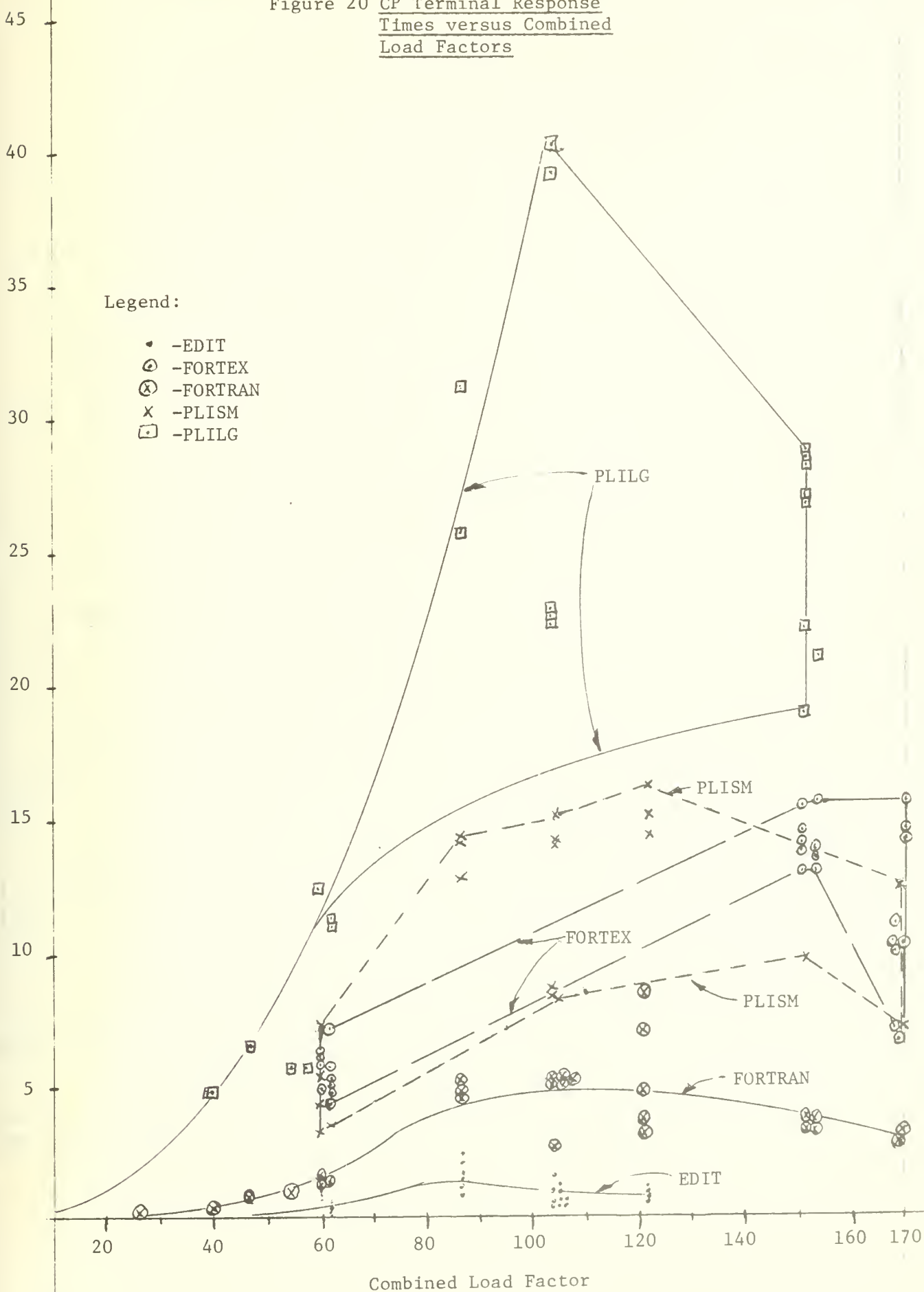


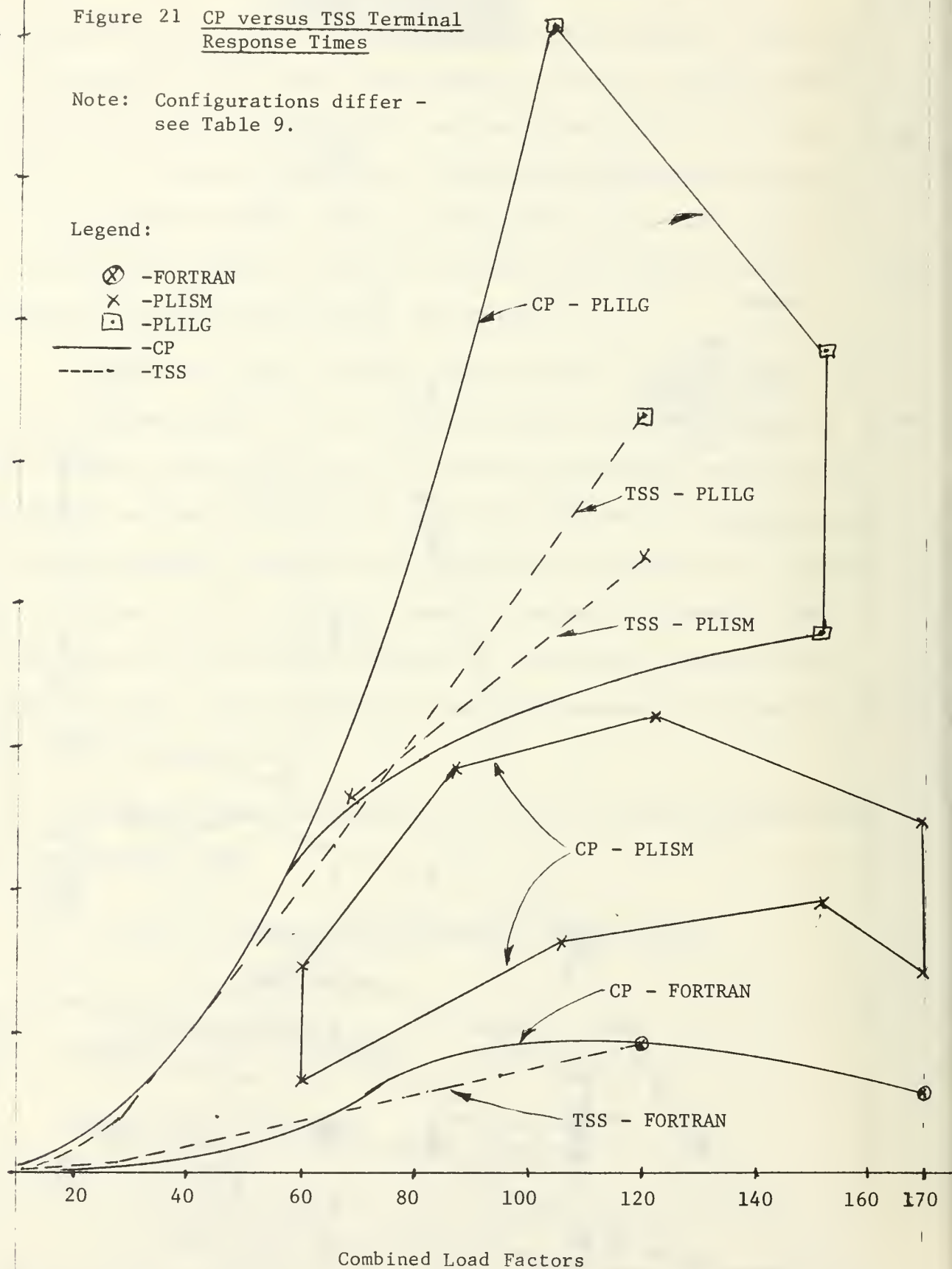
Figure 21 CP versus TSS Terminal
Response Times

Note: Configurations differ -
see Table 9.

Legend:

- ⊗ -FORTRAN
- × -PLISM
- -PLILG
- -CP
- - - -TSS

Terminal Response Times (minutes)



Thus CP has only one third as much memory as TSS; and also it has only a single processor and slower disk units. In time sharing operations, where memory is usually the critical resource, the differences in the configurations is very significant and places CP at a serious disadvantage. Yet CP's performance is competitive.

In order to determine the amount of performance improvement with extra memory, run R31 was repeated as run R36 with the same benchmark but with the two core boxes (512K bytes) configuration. The results showed that CP response times were reduced in half with the extra memory as shown in Figure 22. Thus CP with two core boxes provided twice the performance as TSS with three core boxes*.

6.4.2 Terminal Throughput Comparisons

The calculated terminal throughputs for CP and TSS are shown in Figure 23. The CP throughput data is a replot of that in Figures 7 and 8 but against the combined load factor rather than the CP load factor. The TSS data is simply the curves from Figure 18. The curves show that the CP FORTRAN throughput is better than the TSS equivalent for all load factors less than 70 and about the same for higher load factors. The PLILG throughput is about the same for CP and TSS for all load factors. The measured terminal throughput is essentially the same as the calculated throughput and thus is not presented.

*Actually a test with CP and 3 core boxes was attempted but it had to be terminated prematurely and the results are not considered reliable enough to be included here. However, intermediate results are reported in Table C10 in Appendix C.

Figure 22 CP versus TSS Terminal
Response Times for Three
Configurations

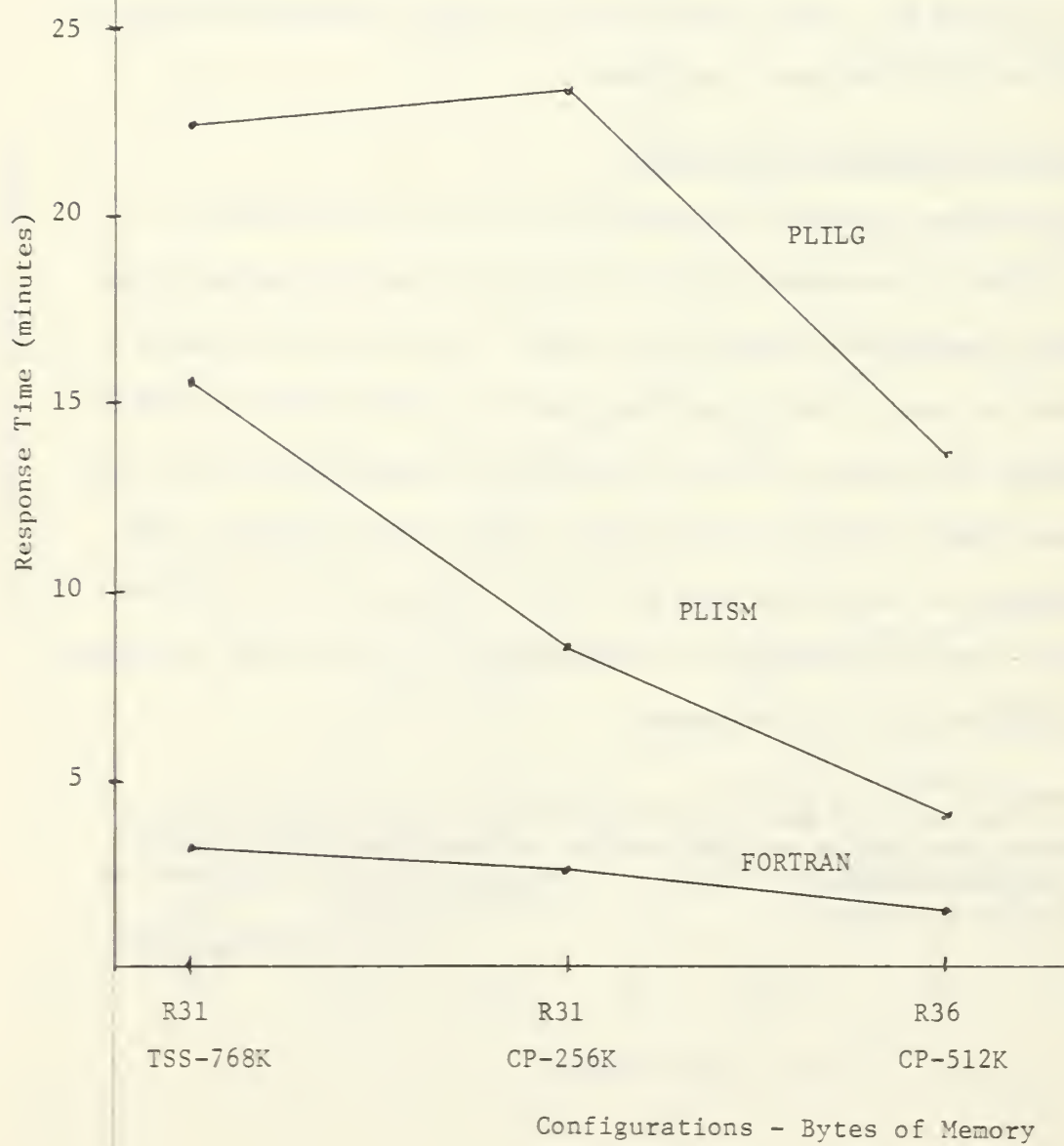
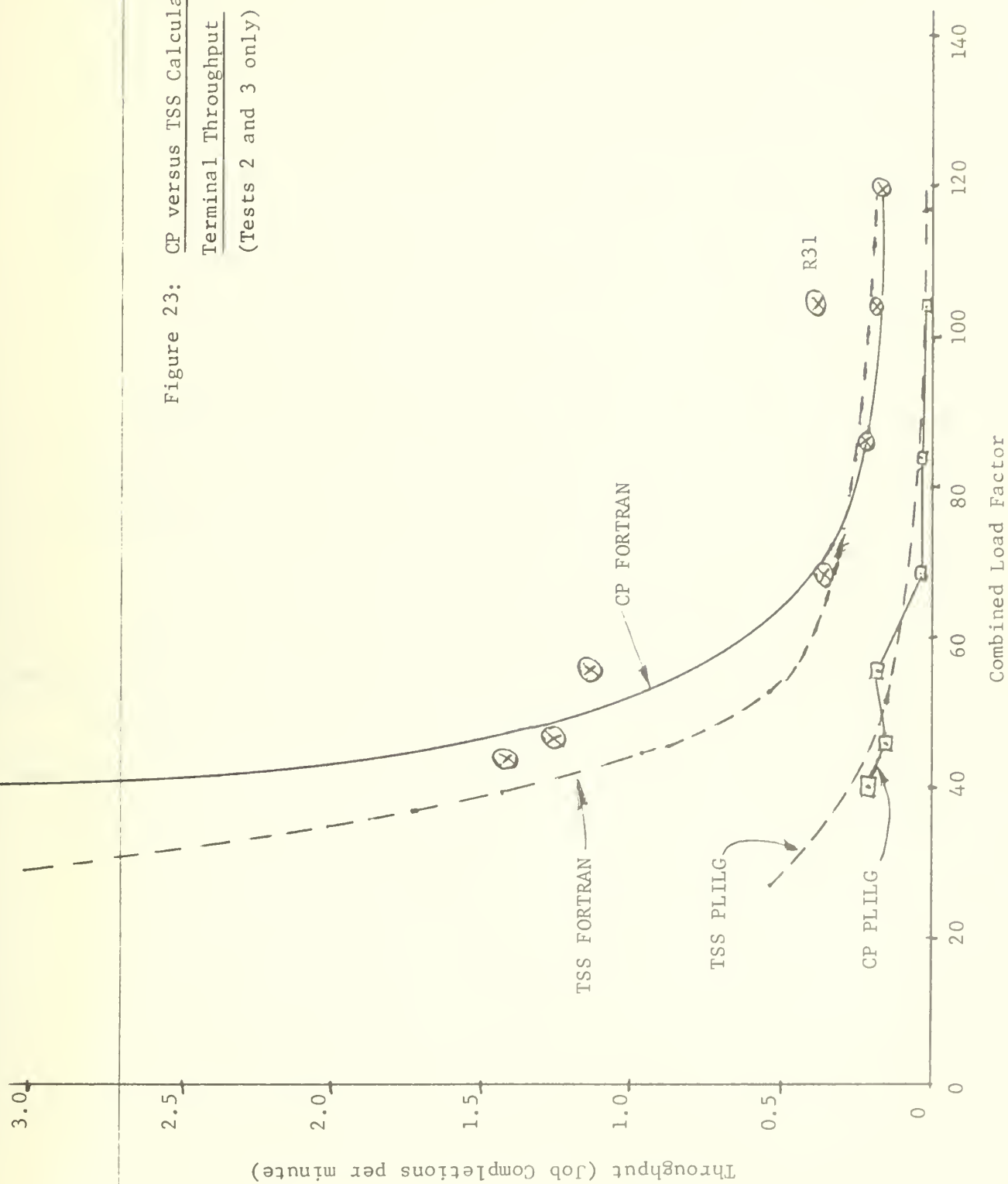


Figure 23: CP versus TSS Calculated
Terminal Throughput
(Tests 2 and 3 only)



Again this shows CP performance is comparable to TSS despite the large discrepancy in resources available (as shown in Table 9).

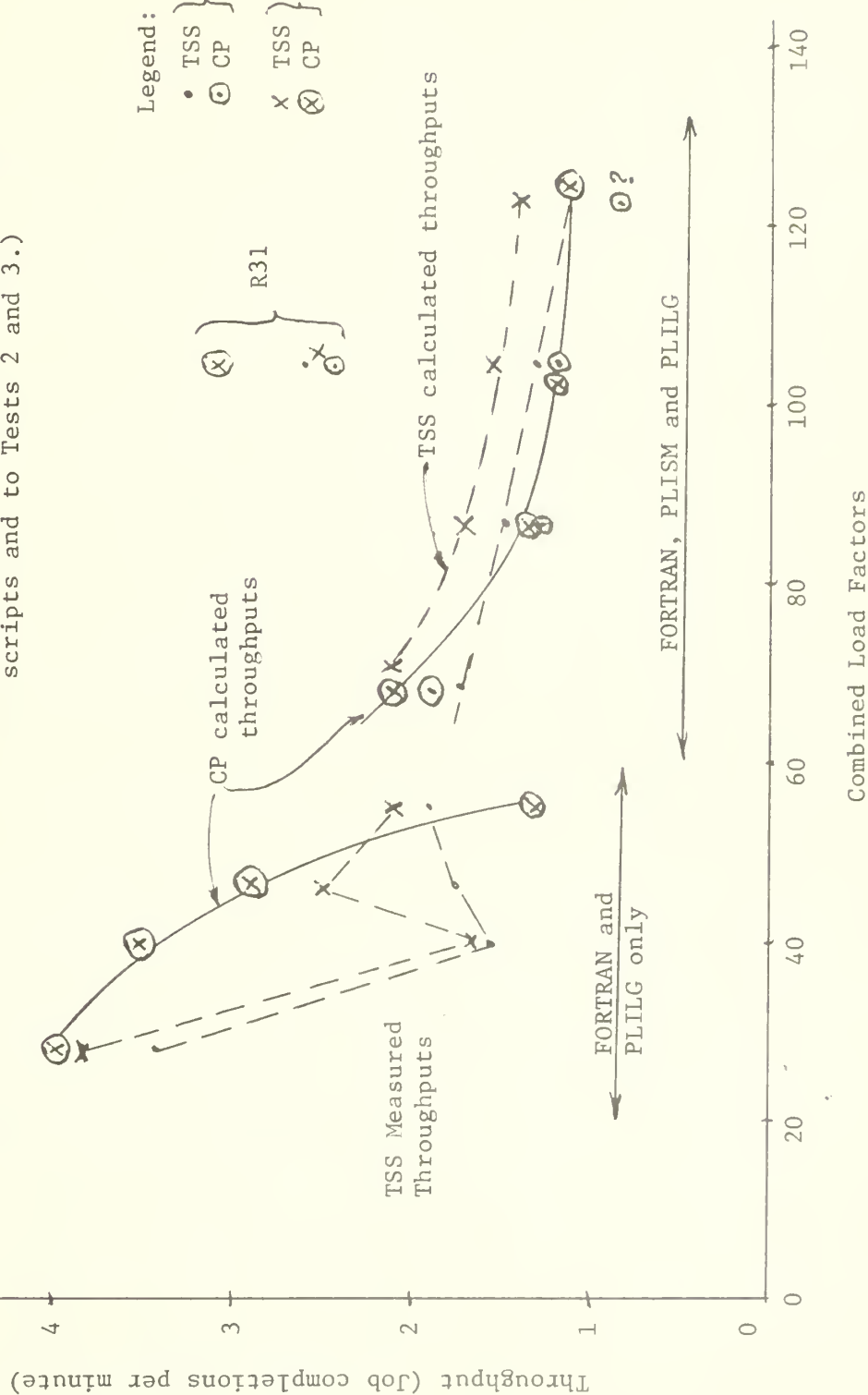
The total throughput for CP and TSS are compared in Figure 24. Since there was only limited throughput data available for TSS, only FORTRAN plus PLILG throughput is shown for Runs R21 to R25 (load factors of 28 to 55), while PLISM throughput is also included in Runs R31 to R35 (load factors above 60). Both calculated and measured throughputs are shown in Figure 24.

Again CP does very well in the comparison of total throughput. Comparing the calculated total throughput, CP has higher throughputs at low load factors and only slightly lower throughputs at the higher load factors.

This completes the comparison of CP and TSS, except for some comparison of all three terminal systems later in Section 6.7. Unfortunately, the effective progress rates for TSS could not be obtained because the stand-alone response times were not available, and therefore no comparisons were possible. Also there were no software monitor measurements for TSS for comparison with the CP data.

Figure 24 Comparison of CP and TSS Total Throughputs

(Total throughput is limited by TSS data available to PLILG, FORTRAN and PLISM scripts and to Tests 2 and 3.)



6.5 MTS Test Results

This section presents the results of 7 MTS runs, which span the range of load used for the CP benchmarking runs. The scripts and programs were the same as described previously. (The listings are also in Appendix A.) The MTS runs were made with the full 3 core-box, dual processor configuration described in Section 1.3.

6.5.1 MTS Terminal Response Times

The MTS terminal response times, as shown in Figure 25, indicate the same general response as experienced in the previous tests. The PLILG responses are the highest, with FORTEX, PLISM and EDIT are next and all about the same, and FORTRAN response times are the lowest. All the response times rise rather slowly as the load increases. For example, the PLILG response times rise from 3 minutes at a MTS load factor of 60 to 7 minutes at a load factor of 130. Suddenly, at a load factor of 130, the response time jumps to 38 minutes. An attempt to explain this peculiarity will be made later in this section.

The only other peculiarity in the above graph is that the EDIT response times seem to be much too high. Actually this is the result of the design of the MTS EDIT script and the operation of the MTS editor. The MTS EDIT script was designed to print an extra ten lines as a means of simulating the one minute delay between editor calls. Also the MTS editor prints every line that it references and thus MTS printed 9 lines (not counting the 10 above) compared to 1-1/2 lines for CP. Therefore, the MTS EDIT response times are almost 2 minutes longer than those experienced in the previous tests.

In an attempt to explain the large jump in PLILG and PLISM response times at a load factor of 130, the following analysis was made. First it should be noted that this jump in PLILG response times from 7 to 38 minutes occurred during a single benchmark test, Run R63. Since it was observed during the test that a terminal executed ~~one~~ PLILG or PLISM script quite rapidly and then took "forever" to complete the second execution, it was decided to plot all the response times for Run R63 as a function of the time of day, as shown in Figure E1 of Appendix E. From this plot, it was confirmed that the response times for the first few minutes of Run R63 were much lower than for the remainder of the run. Thus it was decided to break Run R63 into two runs R63a and R63b, with R63a representing the first few minutes and R63b representing the rest of Run R63.

The response times for Runs R63a and R63b are summarized in Table 11. It can be seen that there is amazing consistency between the response times within the newly defined Runs R63a and R63b, but a lot of difference between the response times.

It is almost as if MTS had labelled some jobs as trouble makers and put them on the back of a long queue, which it never got around to serving. Even an explanation of the priveleged and unpriveleged tasks will not account for this large disparity.

In order to compare MTS results with those from other systems, the MTS terminal response times were also plotted against the combined load factors as shown in Figure 26. The response time curves in Figure 26 are quite similar to those in Figure 25.

Table 11 Breakdown of Terminal Response
Times for Run R63

Run	PLILG		PLISM	
	R63a	R63b	R63a	R63b
Terminal Response Times	8.0	37	3.5	11.7
	8.0	34.5	2.94	10.0
	7.3	28.3		11.5
	12.0			
	8.6	37.22		
	6.7	37.1		
	8.0	36.9		
	9.8	34.0		
Mean	8.55	35.0	3.22	11.1
Variance	19.36	63.09	.156	1.73
Standard Deviation	4.4	7.94	.396	1.31

Figure 25 MTS Response Time for Various Loads

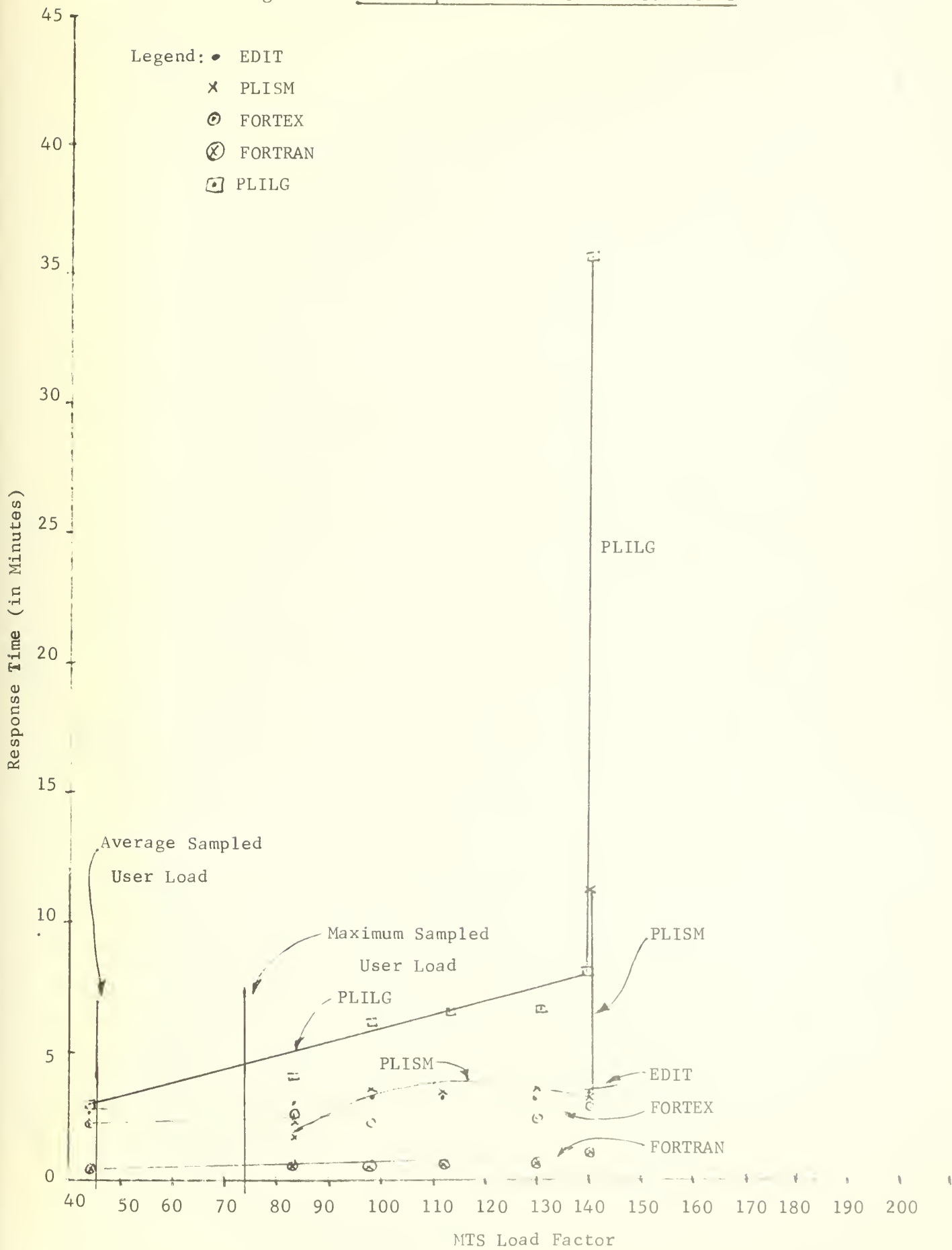
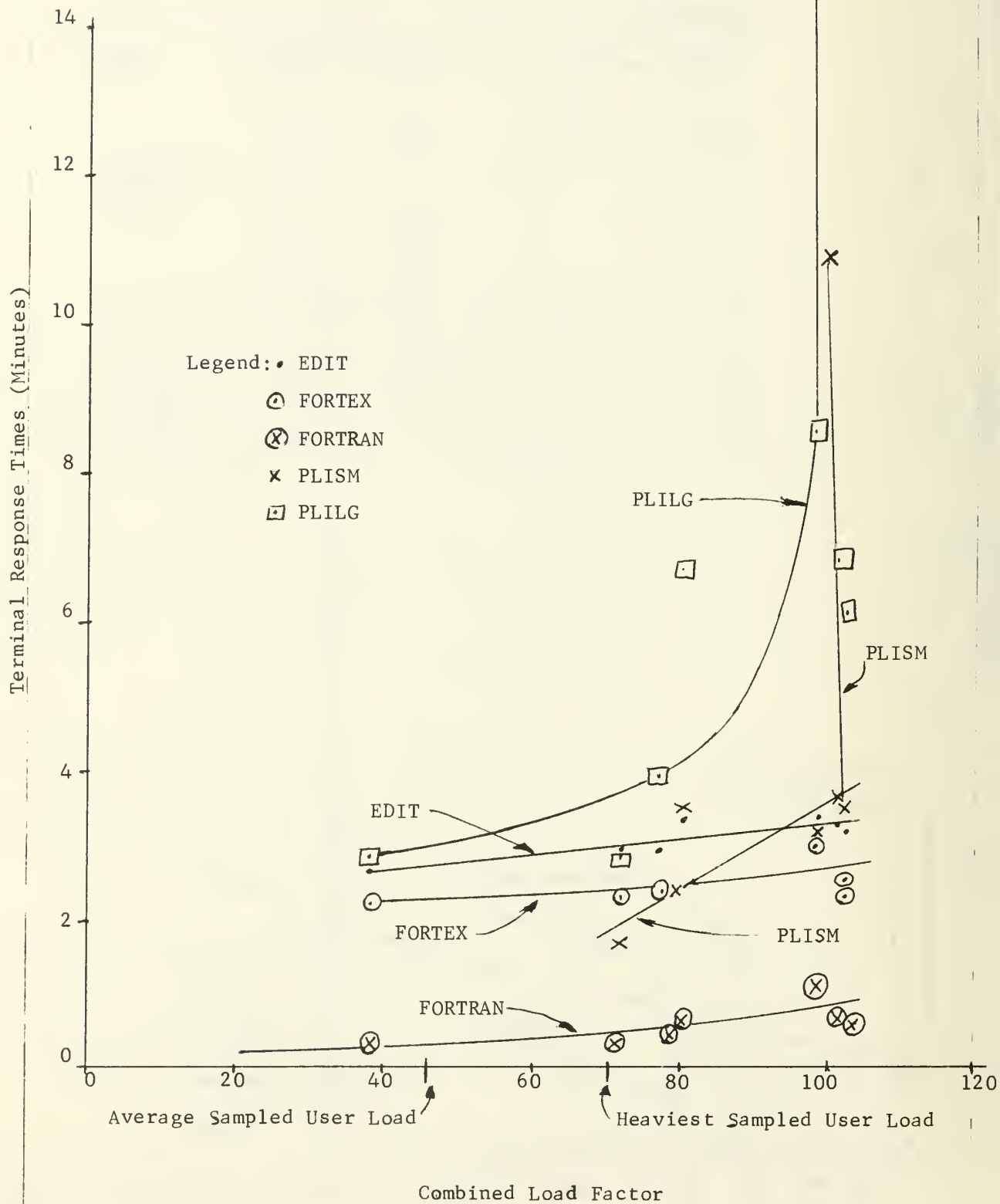


Figure 26: MTS Terminal Response Times versus Combined Load Factor.



6.5.2 MTS Throughputs

The MTS measured terminal throughputs for various MTS load factors are shown in Figure 27. The FORTRAN measured throughput curve has quite a strange shape - almost a saw-toothed shape. It starts at 0.6 job completions per minute at a load factor of 40, increases to 1.8 jobs per minute at 100 and then drops to 0.4 job completions per minute at a load factor of 140. The other measured throughputs are all much less than the FORTRAN throughputs and therefore are replotted on an expanded scale on Figure 27b.

The PLILG measured throughputs drop the most rapidly from 0.3 job completions per minute at a load factor of 40 to 0.03 job completions per minute at a load factor of 140. The PLISM throughputs have a corresponding drop, but the EDIT and FORTEX scripts have a relatively small decrease in throughput as the load increases.

The total measured throughput for all terminal running a particular script is shown in Figure 28. Throughputs for PLISM, PLILG and FORTEX scripts are shown in Figure 28a while those for FORTRAN and EDIT scripts are shown on a larger scale in Figure 28b. As seen previously during the CP test results, the total throughputs per script varies radically as the resource allocation is shifted by the various benchmarks. For specific values or range of values for the throughput see Figures 28a and 28b.

The calculated throughputs for MTS are not presented here because of their similarity with the measured throughputs. This observation is

Figure 27a: MTS Measured Terminal Throughput

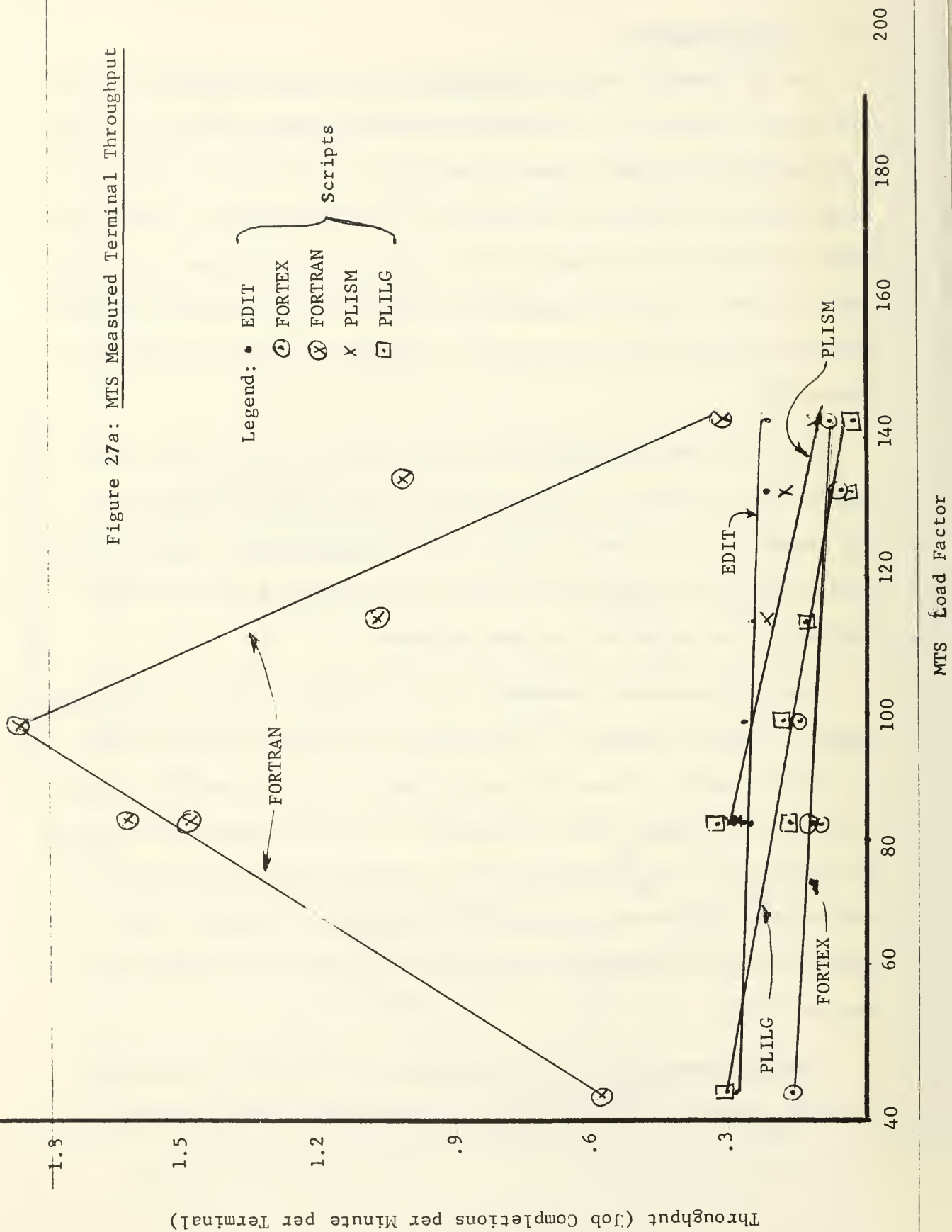


Figure 27b MTS Measured Terminal Throughput - Expanded Scale

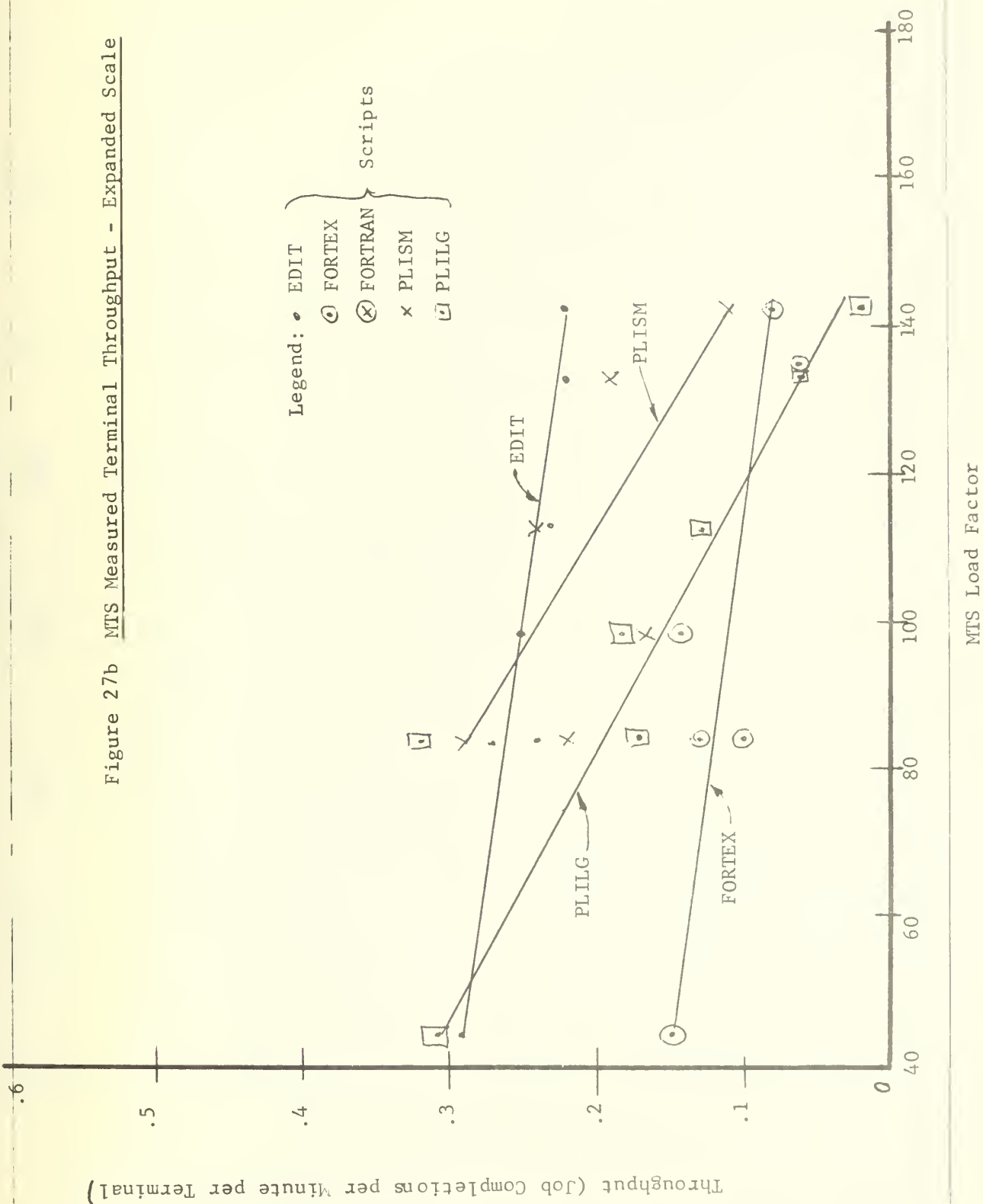


Figure 28a: MTS Total Measured Throughput for
each Script - PLILG, FORTEX, PLISM

(Total measured throughput for all
terminals running a particular script.)

Legend: $\odot$ FORTEX
x PLISM
 $\square$ PLILG

Scripts

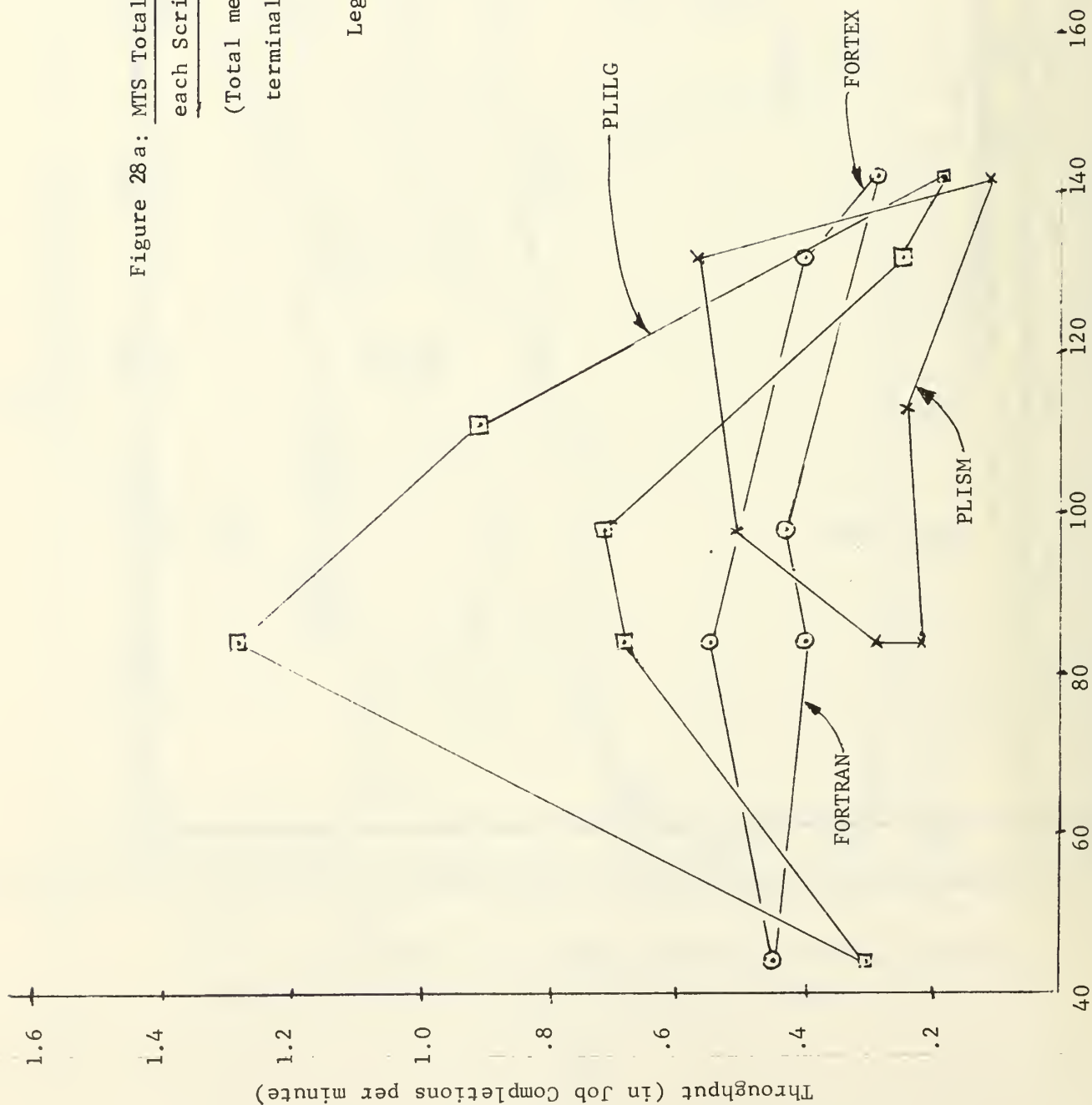
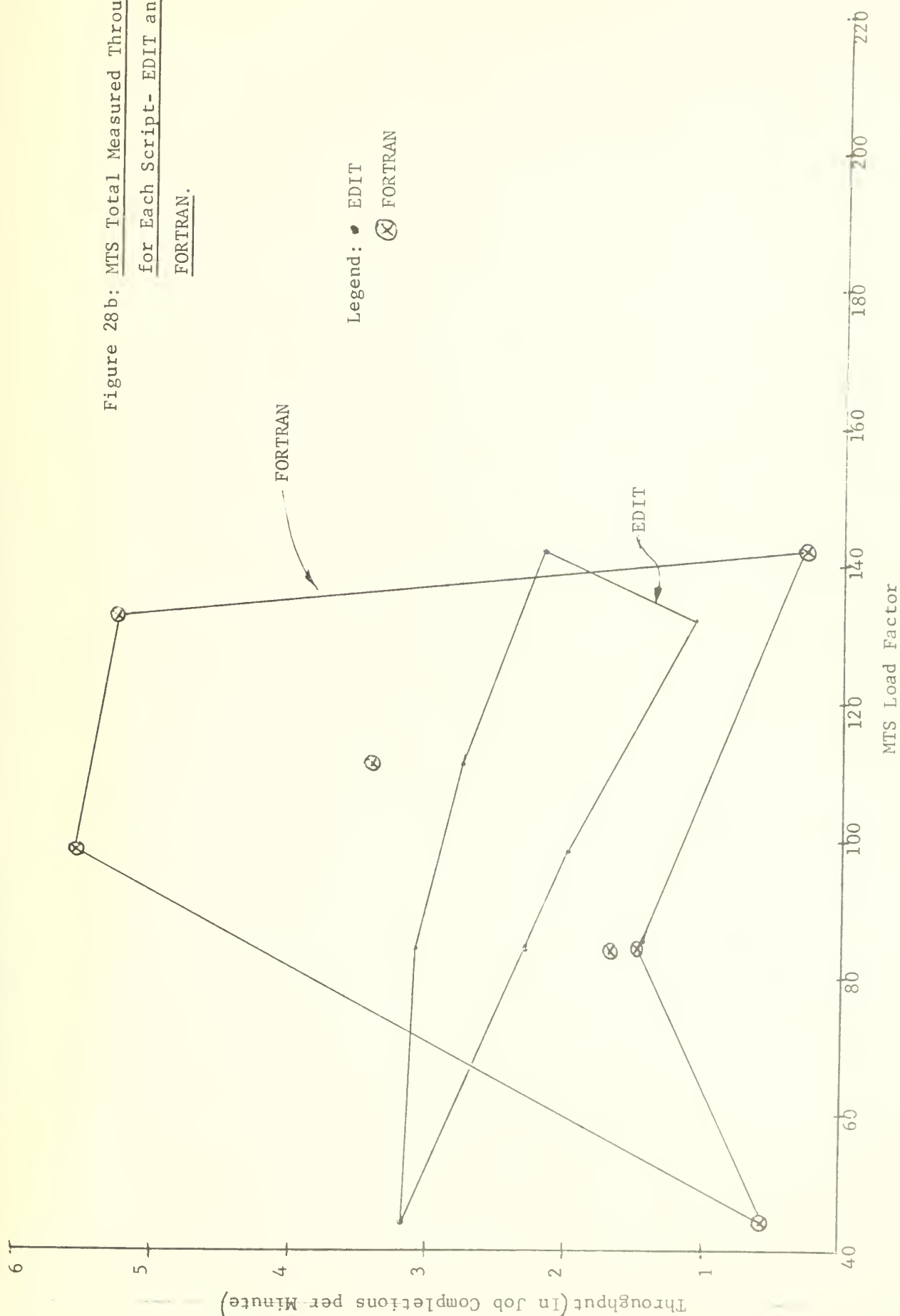


Figure 28b: MTS Total Measured Throughput
for Each Script- EDIT and
FORTRAN.



also consistent with those of previous tests on the other systems.

The MTS total measured throughput ranges from 14 job completions per minute at a load factor of 45 to 6 job completions per minute at a load factor of 140, as shown in Figure 29. The total throughput was subdivided into total EDIT throughputs and total non-EDIT throughputs which are also shown in Figure 29.

6.5.3 MTS Effective Progress Rates

The effective progress rates for MTS were obtained by dividing the mean terminal response times shown in Tables E1 and E2 by the stand-alone response times shown in Table E5. The MTS effective progress rates for each script is shown in Figure 30. The effective progress rates for EDIT and FORTEX scripts is almost constant at 0.75 and 0.1 respectively. The FORTRAN, PLISM and PLILG effective progress rates all decrease rapidly as the load increases and all are approximately the same. They start at about 0.7 for loads from 40 to 80, drop to 0.3 at 100 and then to 0.2 to 0.1 at load factors of 145.

The total effective progress rates and the subtotals for EDIT and non-EDIT scripts are shown in Figure 31. The total effective progress rate appears to have a hump at a load factor of 80 reaching a maximum value of over 12. In all cases the total effective progress rate was over 7 - meaning that the MTS system was operating at least 7 times as effective as a serial processor system.

Of course a large portion of the total effective progress rates was due to the EDIT scripts - anywhere from 3 to 9.5. However the

Figure 29: MTS-Total Measured Throughputs

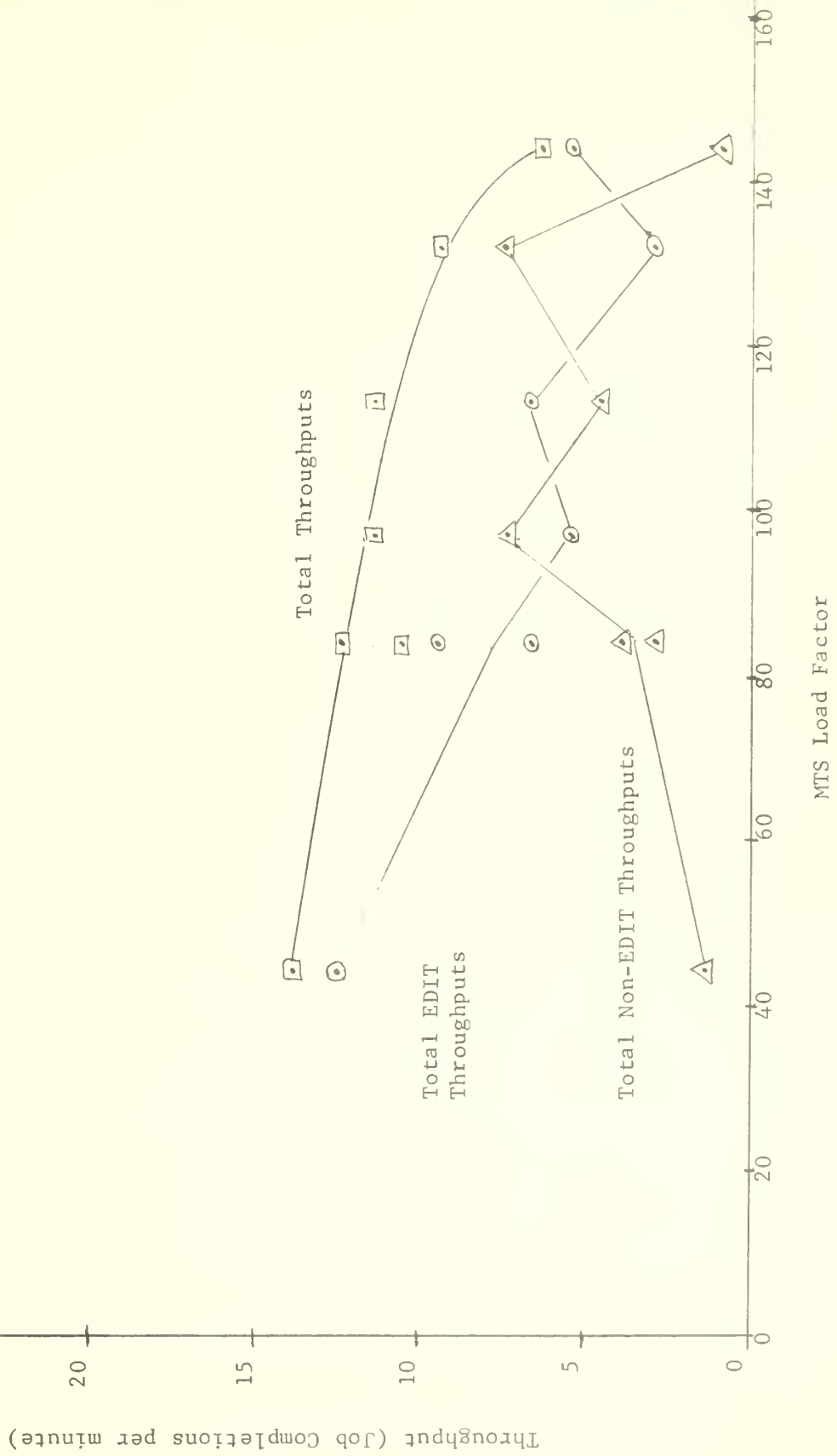


Figure 30: MTS Effective Progress Rates
per Script

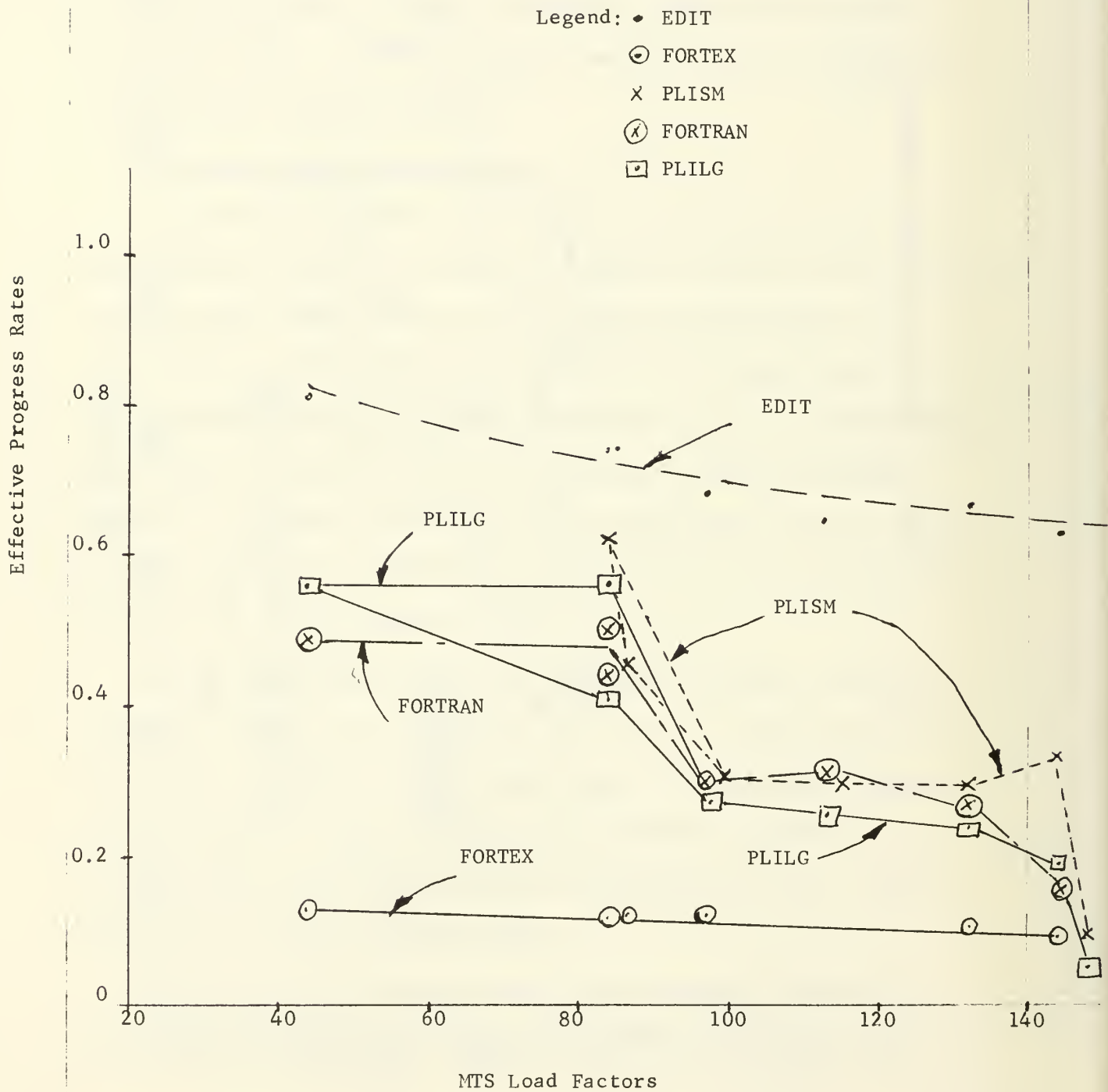
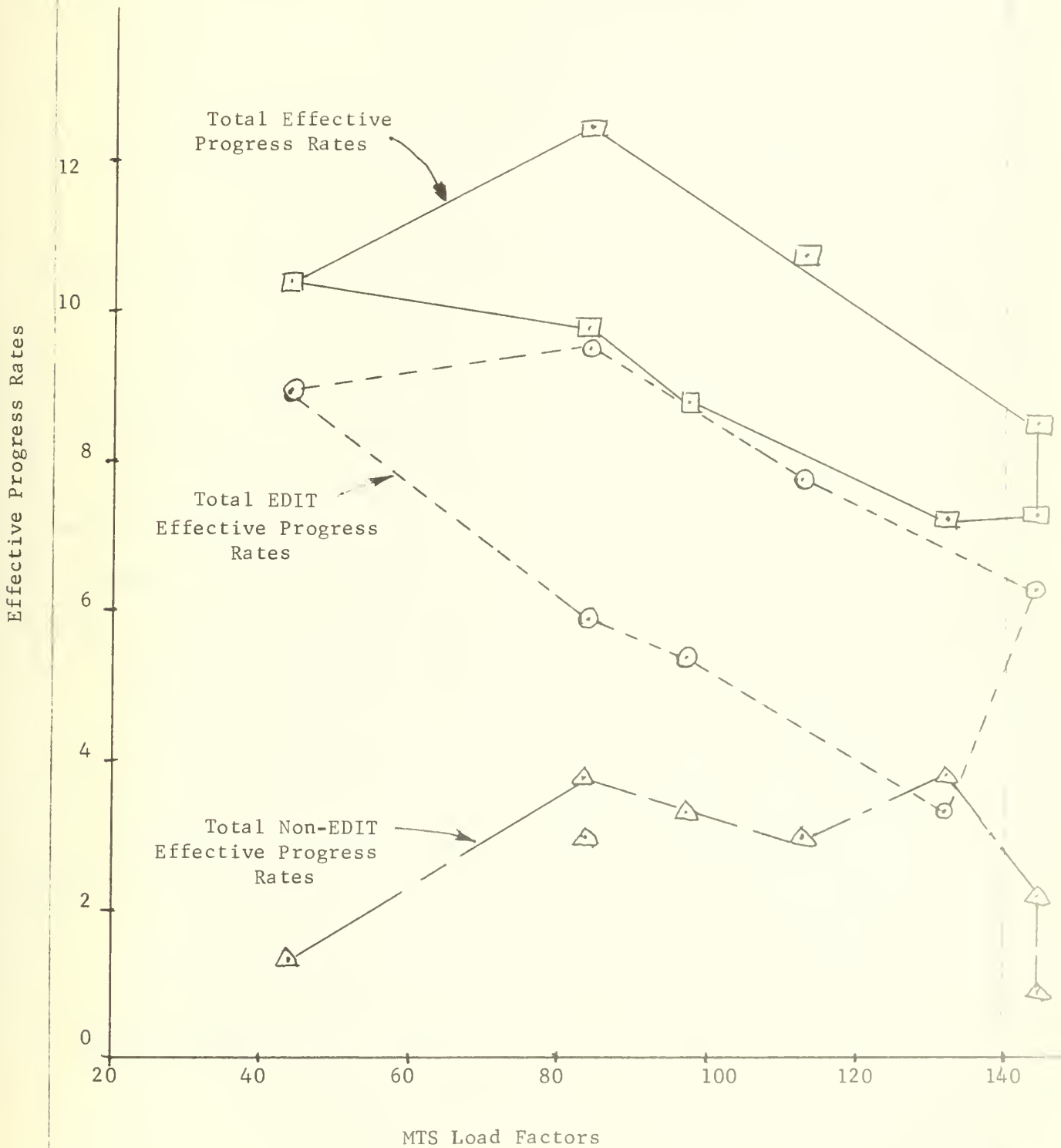


Figure 31: MTS Total Effective Progress
Rates versus MTS Load Factor



effective progress rate for non-EDIT scripts was also impressive, ranging from 1.5 at a light load to 3.9 at a heavy load. Except for the last part of Run R63 when the effective progress rate for non-EDIT scripts dropped to 0.94, the MTS system always operated better than a sequential processor for all loads, even when the EDIT scripts are not included in the total throughput.

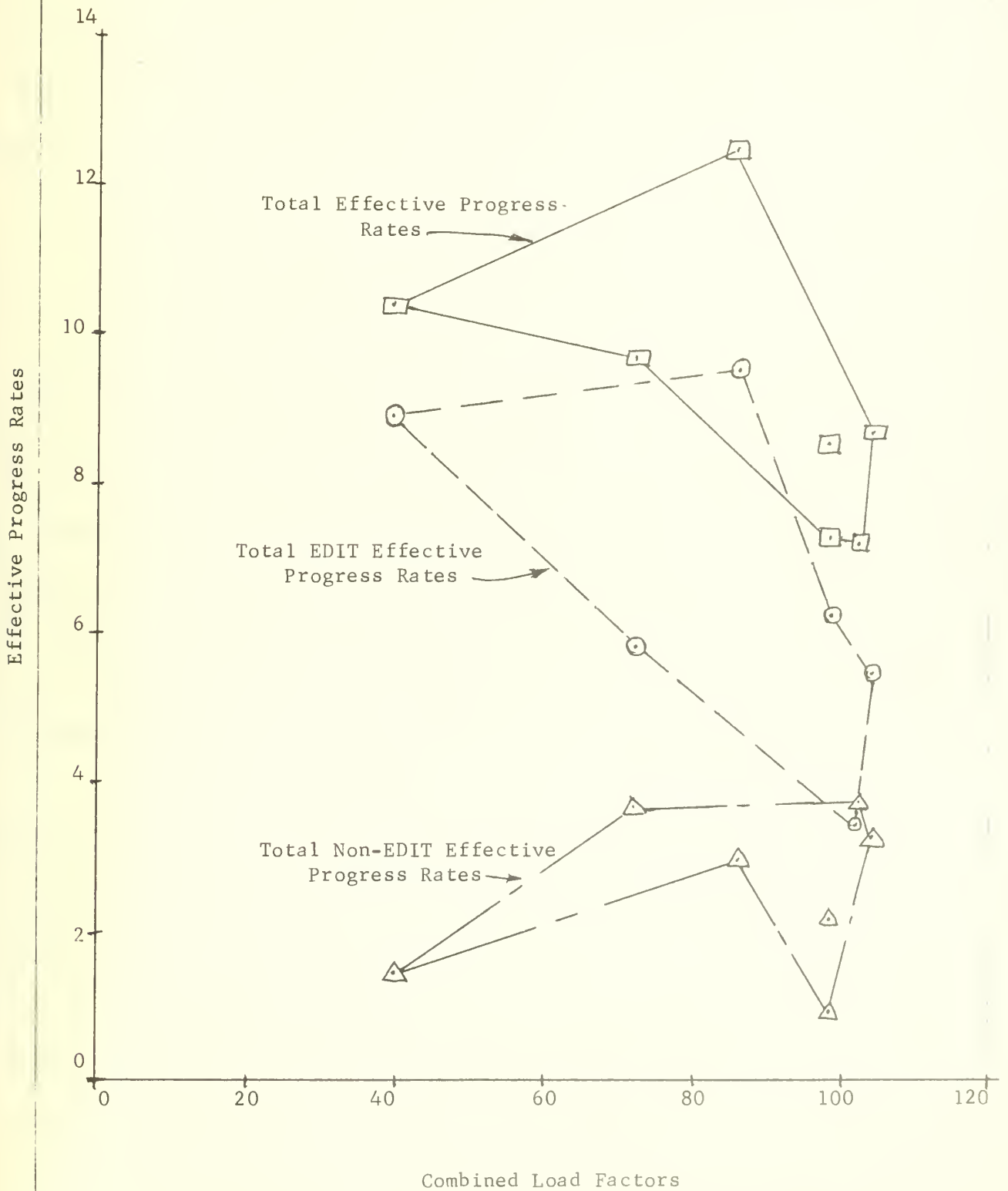
Figure 32 shows the same total effective progress rates as Figure 30, except they are plotted against the combined load factors. This figure will be used in the next section for comparison with CP performance.

6.6 CP and MTS Comparisons

This section compares the performance of the CP and MTS systems as measured in this project. Since one of the objectives of the project was to compare CP and TSS in the configurations at NPS, the CP tests used a different configuration than the MTS tests. To try and compensate for this disparity, the CP and MTS performances are sometimes compared directly, and sometimes compared with CP's performance improved by a factor of three. Since it was shown in subsection 6.4.1 that CP performance doubles when the amount of core memory is doubled, it is assumed that three would be a reasonable CP improvement factor when the amount of core memory is tripled. (The intermediate results in Table C10 of Appendix C also support this improvement factor, although it may be a little optimistic towards CP.)

The performance of CP and MTS was compared on three criteria: terminal response times, total throughputs and effective progress rates.

Figure 32: MTS Total Effective Progress Rates
versus Combined Load Factors



6.6.1 CP and MTS Terminal Response Time Comparisons

A comparison of the terminal response times for each script for CP and MTS is shown in Figure 33. Figure 33 is a composite of the CP response curves from Figure 2 and the MTS response curves from Figure 25. The MTS response times were much lower than the CP ones with two exceptions - one was for Run R63b when MTS had very poor response times*, and the other was for all EDIT scripts because of the extra 2 minutes of printing time used by MTS editor.*

Because CP had a serious configuration disadvantage, having only one core box and one processor compared to three core boxes and dual processors for MTS, the comparison shown in Figure 33 are repeated in Figure 34 with the CP response times reduced by a factor of 1/3. (This is equivalent in assuming the CP performance triples when the amount of memory is tripled.) Since the above comparison tends to indicate that CP was loaded more heavily than MTS - which was not the case -, a better comparison is made when the response times are plotted against the combined load factor, as shown in Figure 35, instead of against individual load factors shown above. Figure 35 is equivalent to combining Figures 20 for CP and 26 for MTS.

In both these figures, the MTS response is generally better than that for CP despite the tripling of the CP performance. The MTS responses for PLILG is better than CP's for all except the lightest load and the heaviest overload in Run R63b. The MTS responses for FORTEX and PLISM are almost always less than the lowest CP ones. The MTS FORTRAN responses

*Discussed previously in Section 6.5.1.

Figure 33: CP vs MTS Terminal Response Times
For MTS Various Loads

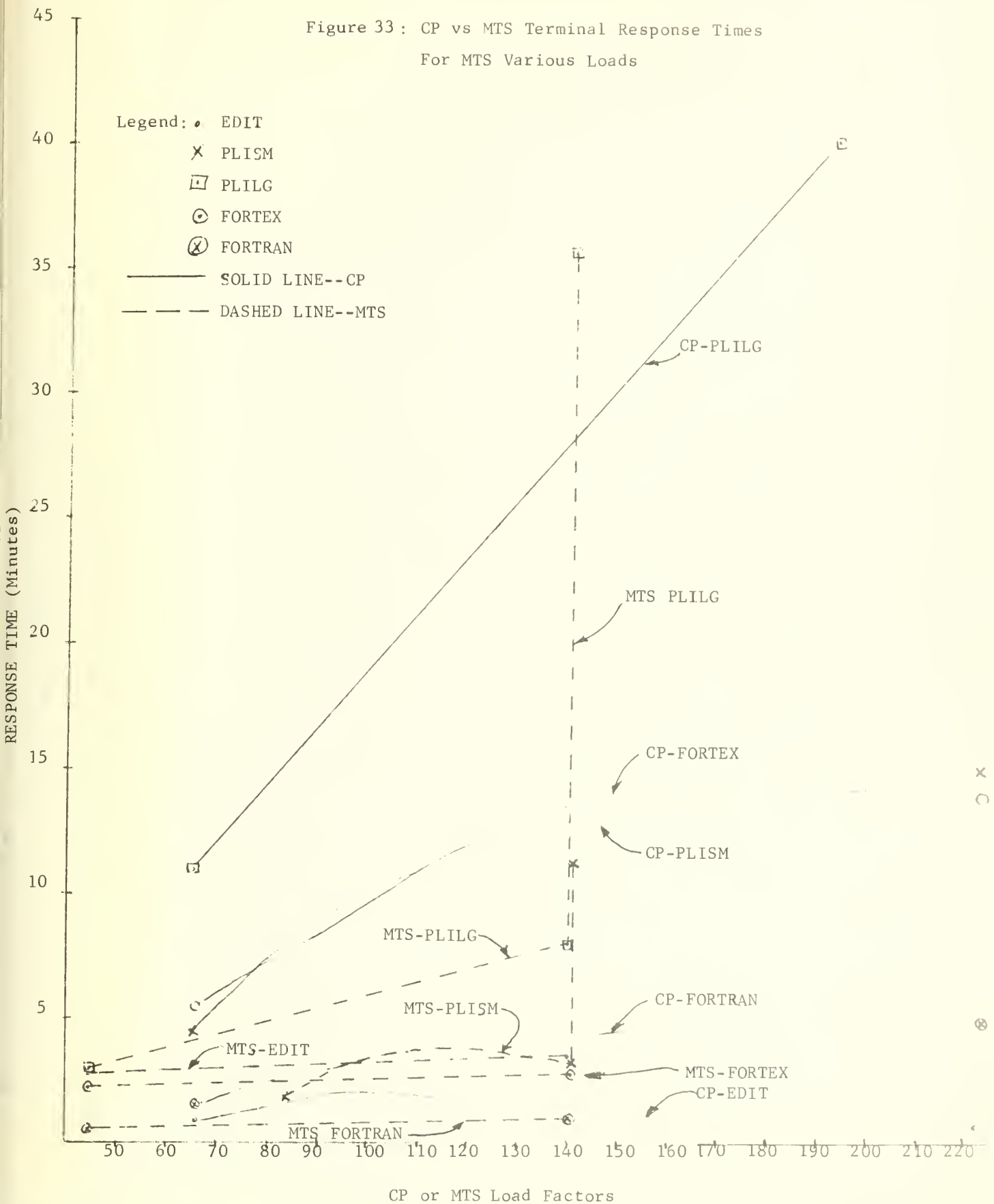
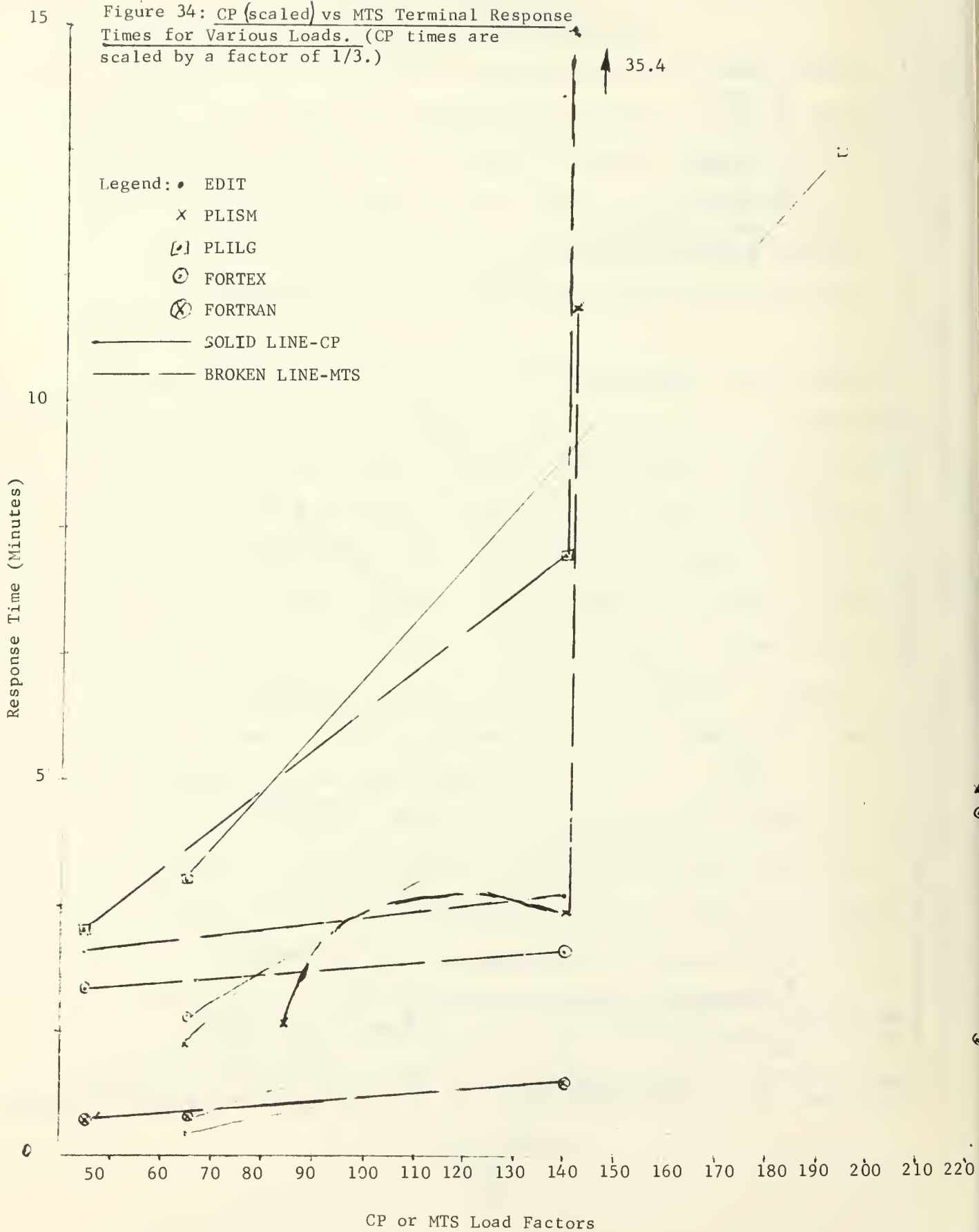
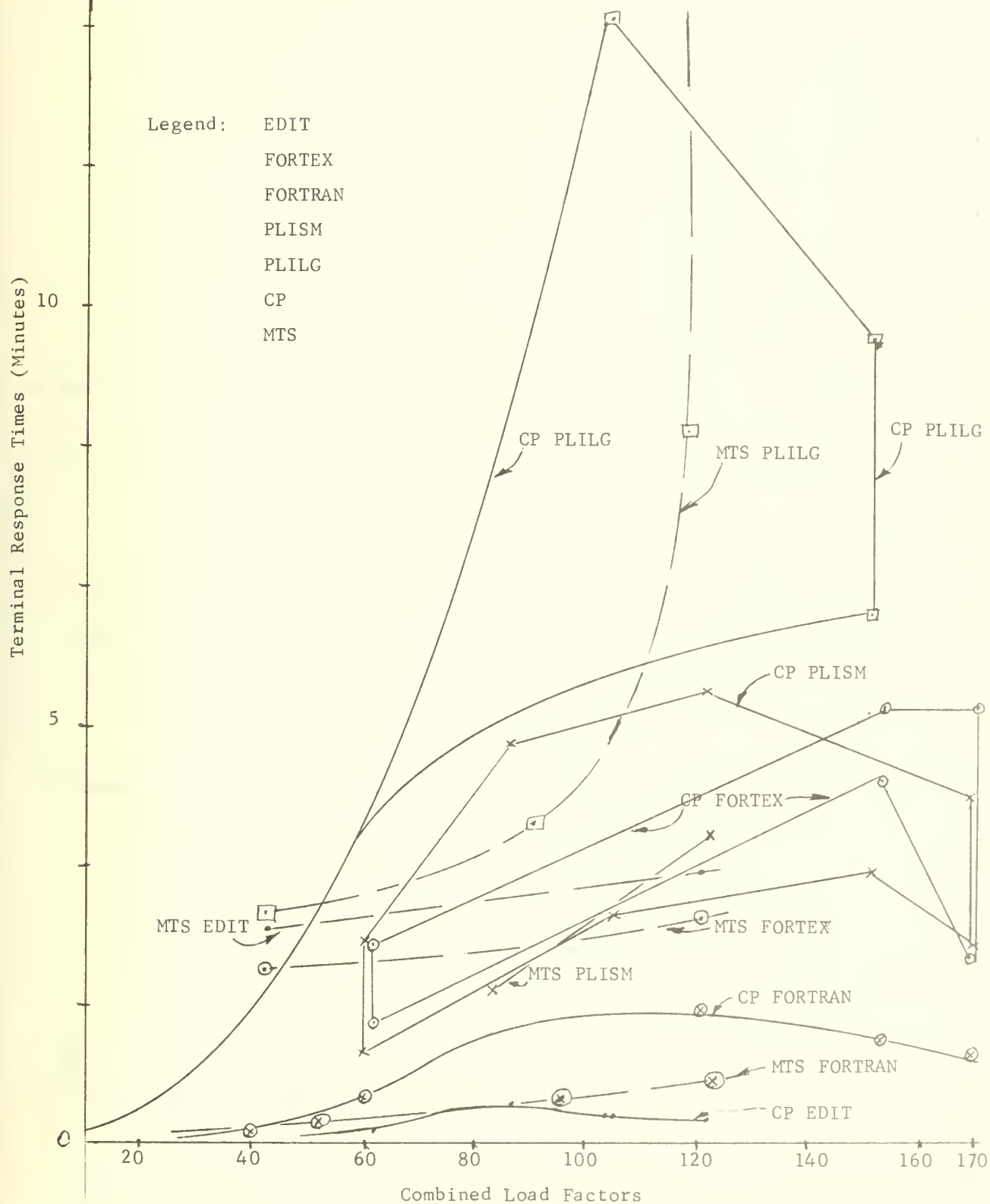


Figure 34: CP (scaled) vs MTS Terminal Response Times for Various Loads. (CP times are scaled by a factor of 1/3.)



15

Figure 35: CP (Scaled) versus MTS Terminal Response Times against Combined Load Factors. (CP times are scaled by a factor of 1/3).



are always much less than the CP equivalents. The only occasion when MTS response times are higher than the CP ones are the MTS EDIT responses, which have the extra 2 minutes of terminal printing times as discussed previously in Subsection 6.5.1.

Another comparison of the CP and MTS performance is shown in Figure 36 in which the terminal response times are plotted against equivalent run numbers. The MTS PLILG response times are lower than the CP scaled response times for all runs except the lightest and the heaviest load. The MTS PLISM responses are somewhat higher than the CP equivalent and the FORTRAN responses are about the same. Therefore, MTS and CP performances are approximately equivalent when the CP response times are reduced by a factor of one-third.

6.6.2 CP and MTS Throughput Comparisons

The total throughputs for CP and MTS are shown in Figure 37, which is a combination of the CP total measured throughput as shown in Figure 9 and the MTS total measured throughput as shown in Figure 29, both replotted against the combined load factor. The MTS total throughput is about double that of CP, ranging from 7 to 14 job completions per minute compared to 3 to 8 job completions per minute for CP. When the CP throughput are multiplied by three, the scaled total CP throughput is larger than the MTS equivalent for all except very heavy overloads (i.e., combined load factor over 100).

6.6.3 CP and MTS Effective Progress Rates

The CP and MTS total effective progress rates are shown in Figure 38,

Figure 36: CP (Scaled) versus MTS
Terminal Response Times for Equivalent
Runs. (CP times are scaled by a factor
of 1/3).

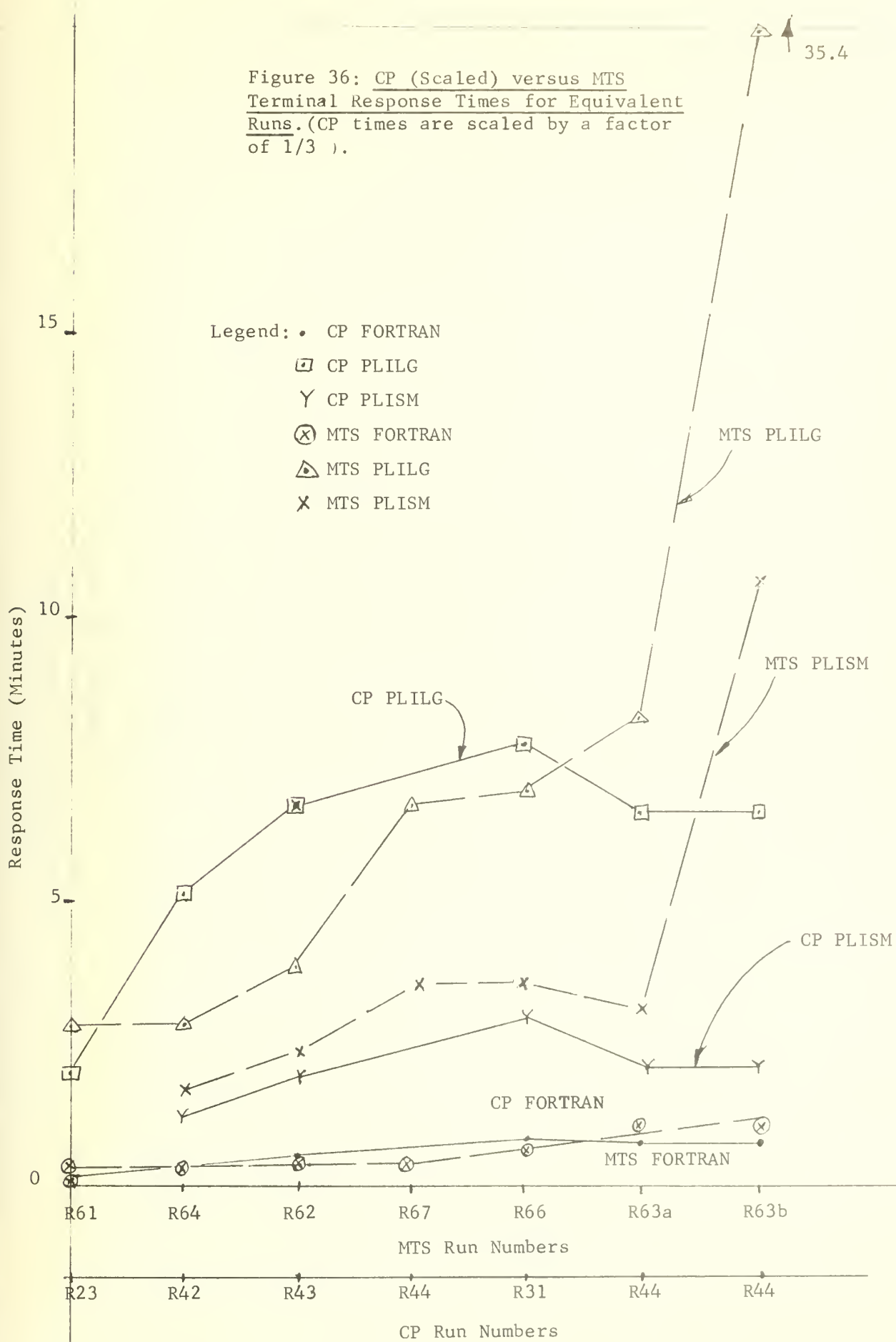


Figure 37: CP and MTS Total
Throughput Comparisons

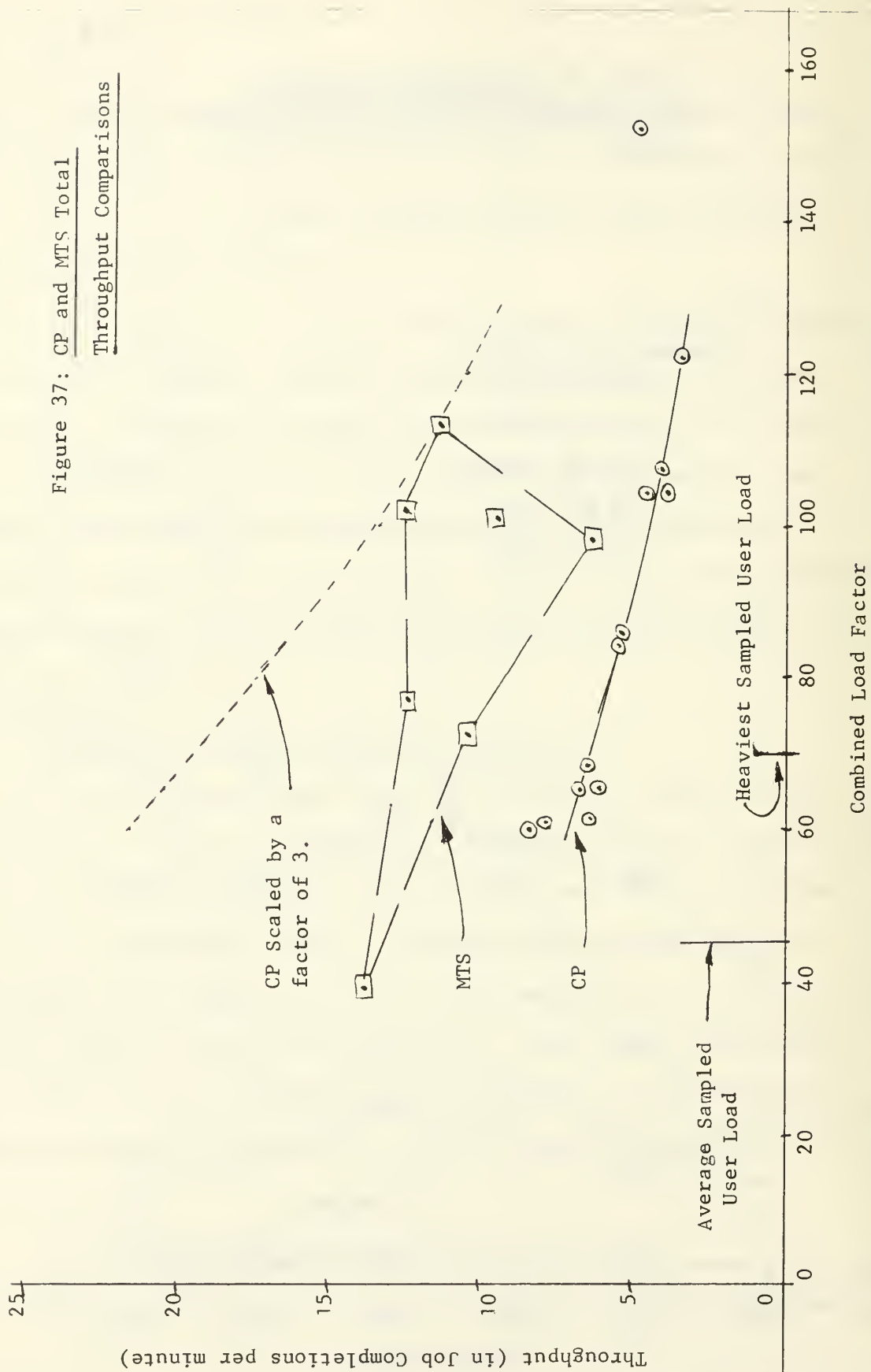
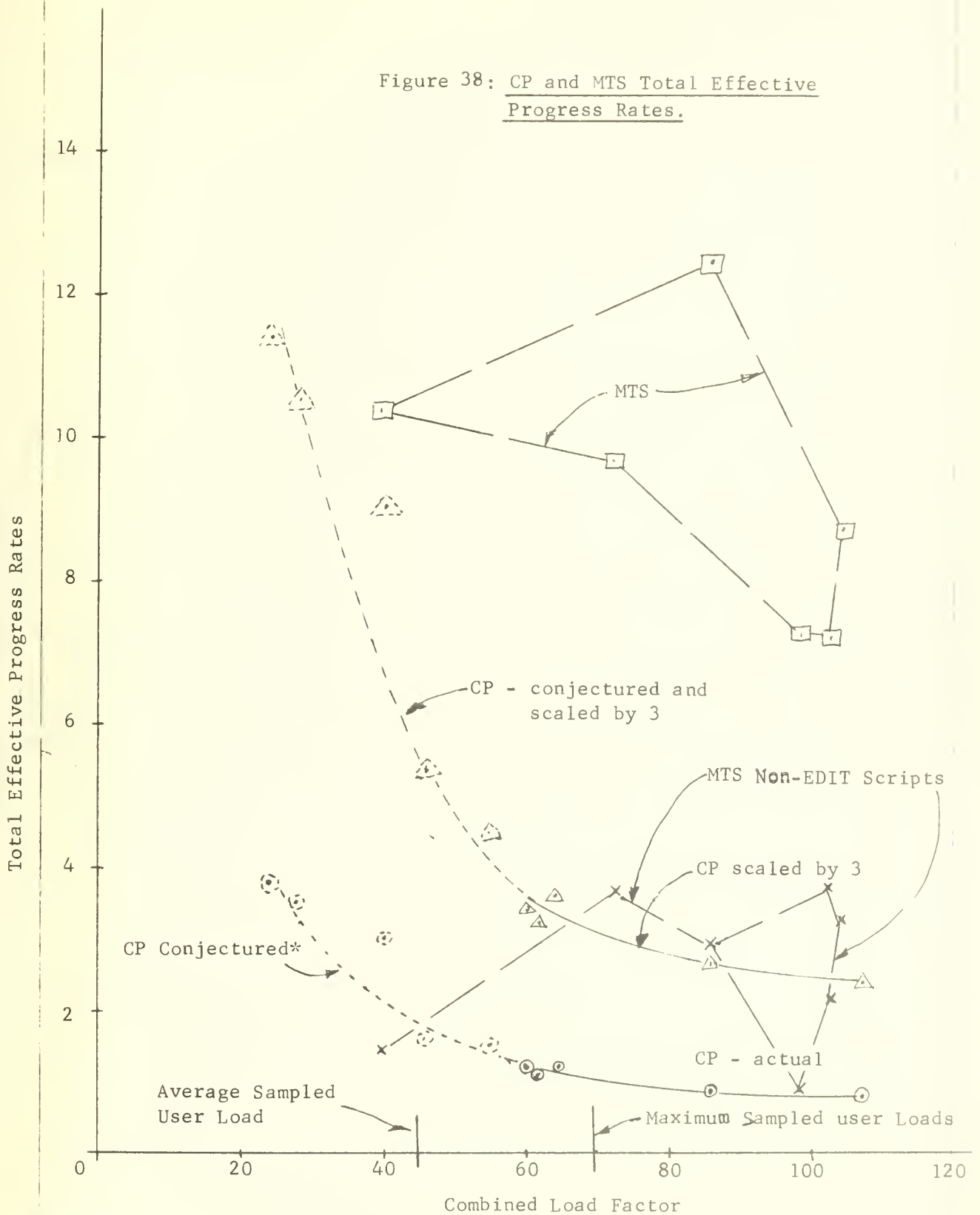


Figure 38: CP and MTS Total Effective Progress Rates.



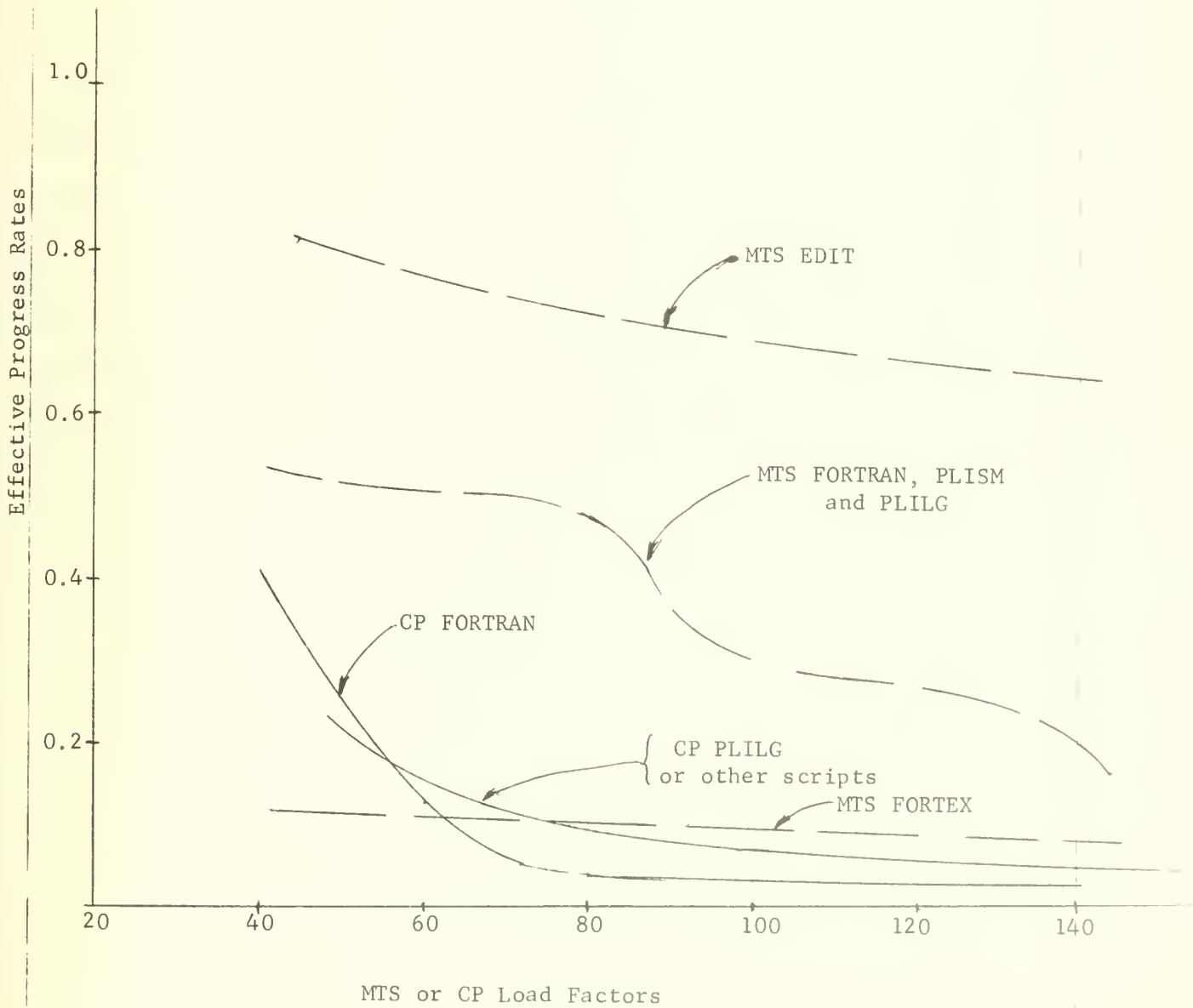
*as conjectured in Table 9 in Subsection 6.2.4.

which is a composite of the CP data in Figure 11 replotted against the combined load factor (actually using Table C7) and the MTS data in Figure 32. The CP data has been augmented at low loads by the conjectured effective progress rates as shown previously in Table 9.

MTS out-performed CP by a wide margin in total effective progress rates. In fact, MTS was substantially better than CP even after the latter's performance was multiplied by three, especially at medium to heavy loads. The MTS total effective progress rates for loads of 60 and above was 10 times better than that of unscaled CP results. (A combined load factor of 60 is below the maximum sampled user load.) Also, the MTS total effective progress rate is 5 times the value for CP at the normal user load. The MTS total effective progress rates for non-EDIT scripts are also shown in Figure 38 and exceed the scaled CP curve on 4 of the 7 runs. A fifth run is very close to the scaled CP curve and the sixth is at a low factor. Therefore, the total effective progress rates for the MTS compilation and execution scripts only is the same or better than that of the scaled CP with all scripts (including EDIT) on almost all the benchmarking runs.

Effective progress rates for some typical scripts for MTS and CP are shown in Figure 39. Again MTS out-performs CP by a wide margin on almost all conditions.

Figure 39: Some Typical Effective Progress
Rates per Script for CP and MTS



6.7 CP, TSS and MTS Comparisons

Rather than repeat many of the figures already presented or variations of them, this section will provide references to similar graphs so that it is easy to make three system comparisons by making pair-wise comparisons.

6.7.1 Pair-Wise Comparisons

The terminal response times for the three systems can be compared by examining the CP and TSS response times in Figure 21 and examining the CP and MTS response times in Figures 34, 35 or 36. Since CP and TSS response times are approximately the same when CP has restricted resources (only 256k bytes of memory), and the CP times scaled by a factor of three are approximately equal to MTS response times, it is easy to conclude that MTS response times are three times better than those for TSS.

The total throughput for the three systems is a little more difficult to compare because of the limited TSS throughput data collected. However the throughput data for the FORTRAN and PLILG scripts can be compared by examining the CP and TSS calculated results in Figure 23 and the MTS measured results in Figure 27. Therefore a comparison of the total throughput of the three systems has to be limited to the comparison of CP and MTS systems as shown in Figure 37.

The effective progress rates of the three systems can not be compared because stand-alone response times are not available for TSS and therefore the effective progress rates for TSS can not be determined.

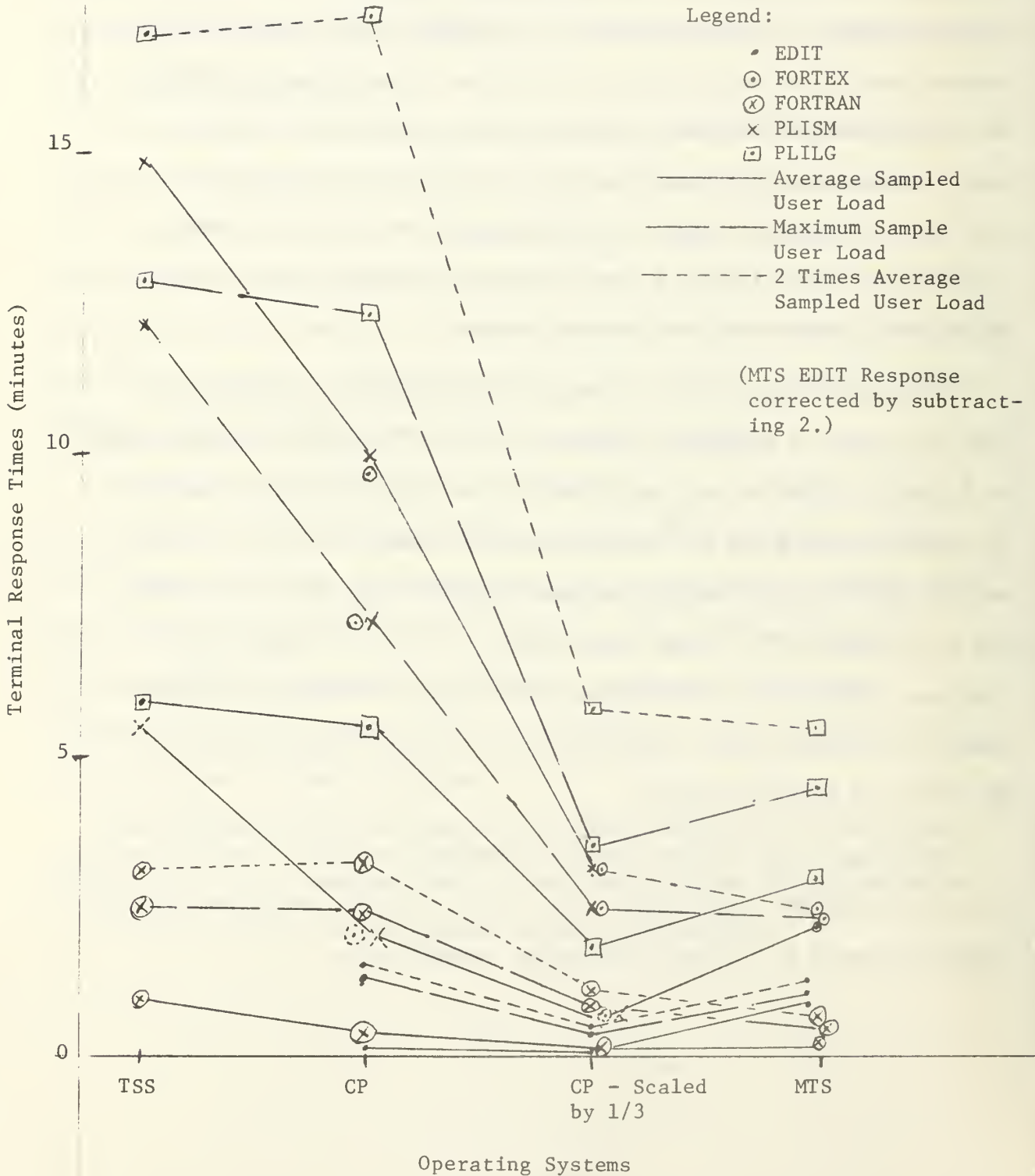
6.7.2 Performance at Typical Loading Conditions

As a summary, the three time sharing systems are compared at typical loading conditions with respect to the terminal response times and the total throughputs. (The total effective progress rates are not compared because there is no data for TSS). The three typical loading conditions are: the average sampled user load and the heaviest sampled user load and a heavy overload which is hereby defined as twice the average sampled user load. These three loads are determined by the terminal probing tests in Sections 5.3 and 5.4, and correspond to CP load factors of 50, 80 and 100, respectively, or the equivalents.

Therefore, Figure 40 is a summary of the terminal response times from the curves in Figures 2, 16 and 25 for CP, TSS and MTS, respectively. At a glance, Figure 40 indicates that TSS and CP provide approximately the same response times at the three selected loading points. Also CP and MTS provide approximately the same response times when the CP times are all divided by 3. On the other hand, the load has some effect on this last comparison. At the average user load, the scaled CP response times are generally better than MTS, but at the overload conditions the MTS times are generally better.

The normal user response times for FORTRAN are 1.0, 0.4 and 0.2 minutes for TSS, CP and MTS respectively. The PLILG response times under similar conditions are 5.8, 5.5 and 3.0 minutes, respectively.

Figure 40: Terminal Response Times for
Selected Load Conditions
for CP, TSS and MTS



The three time sharing systems are also compared with respect to the total throughput. The data for the total measured throughputs for the TSS and CP comparison was obtained from Figure 24, while the throughput data for the CP and MTS comparison was obtained from Figure 37. The summary, as shown in Figure 41, indicates TSS and CP have the same throughput at the heavy load and the overload conditions, but CP has better performance at light loads. The total measured throughputs for CP and MTS are approximately the same when CP's throughputs are multiplied by two instead of three.

This concludes the results and comparisons of the three time sharing systems. The next section compares the three systems with batch handling capabilities under batch benchmark loads.

6.8 Batch Benchmarking Results and Comparisons

This section presents the results of the batch benchmarking tests conducted on the TSS, OS/MVT and MTS operating systems and compares their performances. The batch benchmarking programs or jobstreams were described previously in Subsection 2.2.2 and Appendix B, and the tests performed were described in Section 3.2.

6.8.1 Batch Benchmarking Results

The results of the OS batch benchmarking tests are summarized in Figure 42. The results show that the three jobstreams take about the same total turnaround times on OS, varying from 22 to 27 minutes. Also there is relatively little difference between the results with two and three core boxes - which is very surprising considering the useable core is doubled in the second case.

Figure 41: Total Measured Throughput for
Selected Loads for CP, TSS and MTS

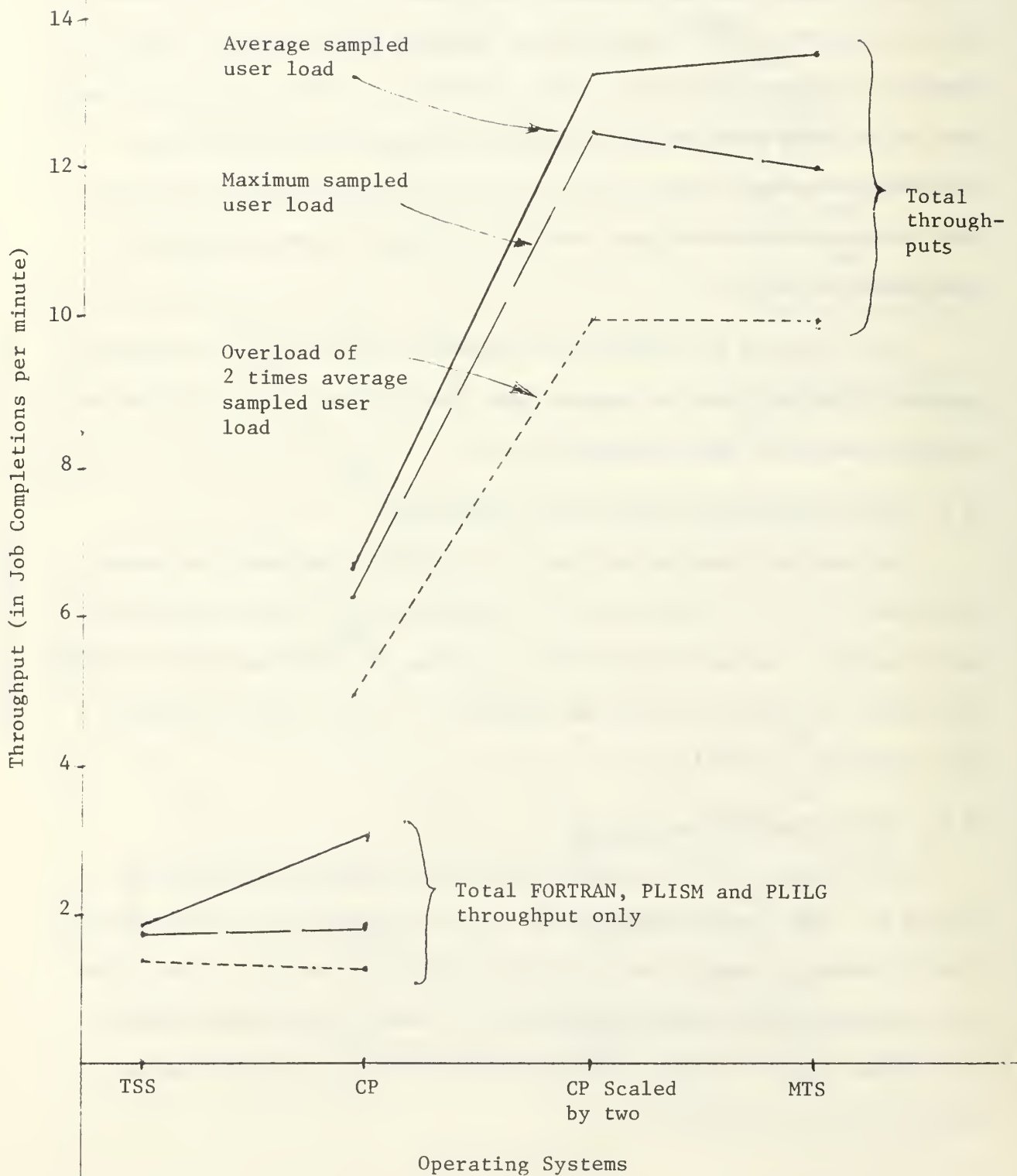
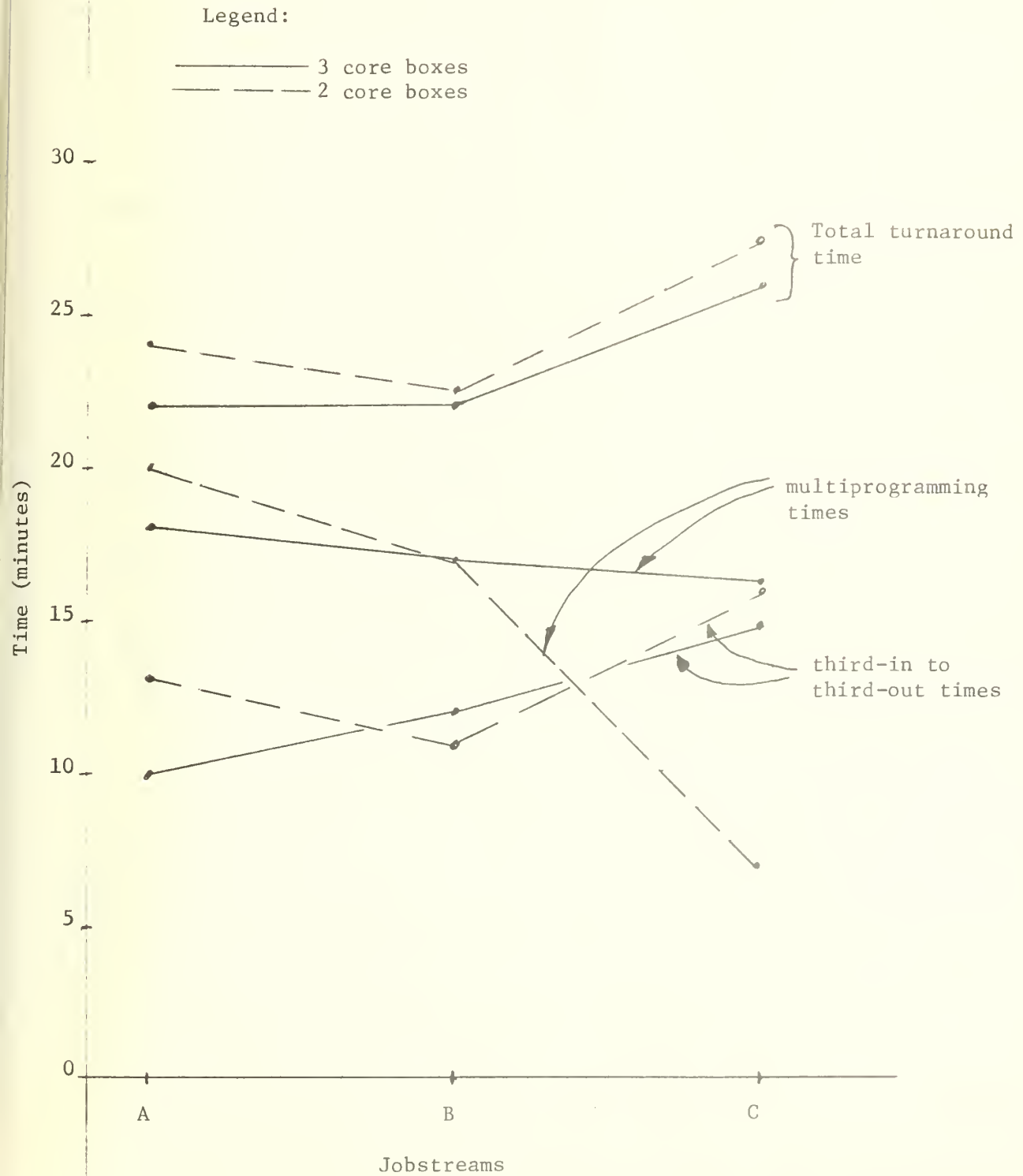


Figure 42: OS Benchmarking Results



The multiprogramming times - the times during which at least two jobs are in execution - vary from 17 to 20 minutes on Jobstream A and B but vary from 7 to 16 minutes on Jobstream C. The very low multiprogramming time with two core boxes indicated a very inefficient use of OS, since OS is operating as a serial processor for 20 minutes out of the 27 minutes. This benchmark must be somewhat atypical when the multiprogramming time can drop from 16 minutes to 7 minutes and the total turnaround time only increases from 26 to 27 minutes.

The third-in to third-out times - which represent the times in which at least three jobs are in the system - are also shown in Figure 42. These times also represent the time in which the system is loaded and vary from 10 to 16 minutes for the three jobstreams and the two configurations.

As an aside, some results from a hardware monitor are shown in Table 12, which indicate that the CPU wait time decreases from 65 percent to 47 percent during normal operation when the memory was increased from 512K bytes to 768K bytes [9]. This means an improvement in CPU utilization of 50 percent (i.e., $(65-47)/(100-65)$) when the extra core box is added. This means a much larger difference in performance with the extra core box would be expected under normal batch operation than that experienced in the batch benchmarking test results above. This suggests that the batch benchmarks do not take advantage of the extra memory available with 3 core boxes.

6.8.2 Batch Performance Comparisons

The comparison of TSS, OS and MTS under batch loading is shown in

Table 12: Hardware Monitor Results for Comparing OS/MVT Performance with 768K bytes vs. 512K bytes of Core under Normal Operating Load

<u>EVENT</u>	<u>512K bytes no drum</u>	<u>768K bytes and drum</u>	<u>Ratios</u>
CPU wait	65.18	47.13	1.38
CPU wait and channel <u>not</u> busy	29.73	26.07	1.14
CPU wait and channel busy	34.61	21.06	1.65
CPU wait, and channel not busy and MVTREX seek	3.51	2.22	1.58
CPU wait and channel not busy and MVTLNK seek	12.34	7.62	1.62
CPU wait and channel not busy and LINDA seek	2.87	0.12	23.9
CPU wait and channel not busy and SPOOL 1 seek	2.26	1.14	1.98
CPU wait and channel not busy and SPOOL 2 seek	6.03	2.15	2.8
CPU wait and channel not busy and SPOOL 3 seek	2.64	2.52	1.05

Figure 43. TSS with two core box simplex configuration had the poorest performance, especially on Jobstream A where it required 41 minutes for completion. On Jobstream C, OS with either 2 or 3 core boxes was almost as bad as TSS at 27 minutes. The OS performance on Jobstreams A and B was about the same as the TSS 3 core box simplex system on all three jobstreams with turnaround times of 21 to 26 minutes. The TSS 3 core box dual processor system performance was significantly better than the OS or other TSS performances, ranging from 20 minutes on Jobstream A to 14 minutes on Jobstream C.

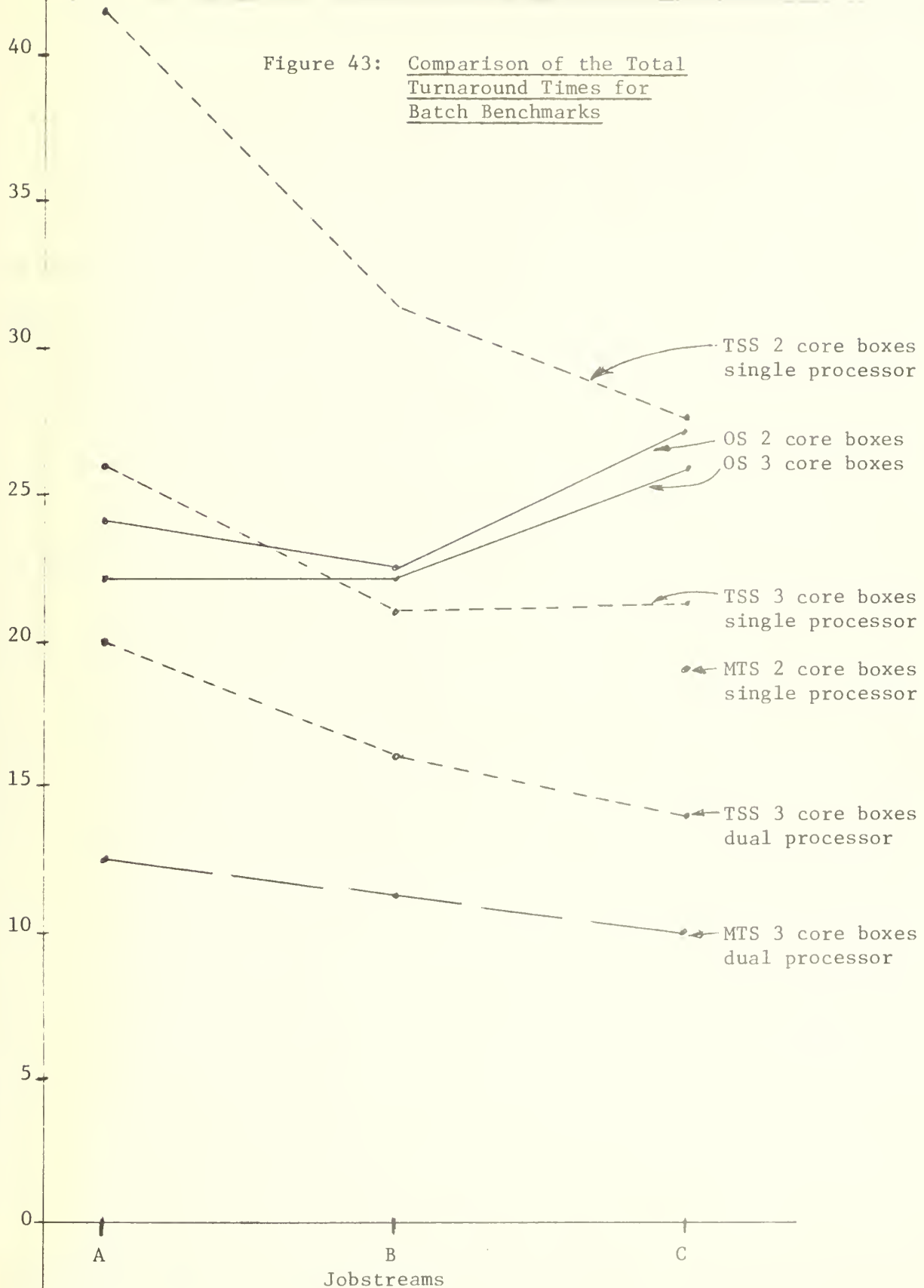
The MTS performances were already superior to all others. The MTS performance with 2 core boxes and a single processor was better than TSS with three core boxes and a single processor on Jobstream C (but was not as good as the TSS duplex 3 core box performance). The MTS dual processor three core box performance was the best of all with turnaround times of 12.5 minutes and 10 minutes for Jobstreams A and C respectively. Compared to OS with compatible configurations*, the MTS performance was 45 percent better than that of OS on Jobstream A and 61 percent better than that of OS on Jobstream C.

Before continuing, one comment on the batch benchmarks is in order. Since Jobstreams A and B contain 2 300K byte jobs and Jobstream C contains 3 of these large jobs, OS must serially process these jobs with either 2 or 3 core boxes. However if the largest jobs had been 250K bytes, then OS would have been able to multiprogram the large jobs when it had 3 core

*OS can not run with two processors

Figure 43: Comparison of the Total Turnaround Times for Batch Benchmarks

Total Turnaround Time (minutes)



boxes, resulting in a considerable improvement in performance.

If the improvement had been the same as that experienced in normal operating conditions as presented in Subsection 6.8.1, i.e., 50 percent, then the OS total throughput times would have been 16, 14 and 18 minutes for Jobstreams A, B and C respectively. This would have meant that with nearly equivalent configurations*, OS would have out-performed TSS on 2 of the 3 benchmarks. TSS and MTS performances would not change appreciably, since they use paging instead of fixed memory allocation. Of course, this comparison is only a conjecture based on the observed improvement in CPU utilization during normal OS operation and not on an analysis of the jobstreams.

*Of course OS would have only a single processor while TSS had 2 processors.

6.8.3 Comparison of MTS with CP Plus OS

Since one of the objectives of this research was to compare the performance of MTS with that of the split CP-OS system, this section presents a tentative comparison of MTS supplying both batch and terminal services to that of the split system with CP supplying the terminal services and OS supplying the batch services. Although the testing of MTS with both batch and terminal services has not yet been accomplished, some analysis based on the results in this report are now presented.

Consider the situation in which the batch benchmarking loads are applied to MTS and, as soon as they are completed, the terminal benchmarking loads are applied to MTS. The question is how many jobs will be completed by MTS compared to the total number completed by both CP and OS during the same time period. Consider the following:

- (1) The time for OS with 2 core boxes to complete all three job-streams, as shown in Figure 43, is 73 minutes ($24 + 22 + 27$).
- (2) The time for MTS with the full system to complete all three job-streams is 34 minutes ($12.5 + 11.3 + 10$).
- (3) The difference between MTS time and OS times is 39 minutes, which is available for processing terminal jobs.
- (4) Since the total throughput for CP with one core box at the average sampled user load is 6.7 jobs per minute according to Figure 41, the total throughput in 73 minutes is 490 jobs (6.7×73).

(5) Since the total throughput of MTS with the full system at the average sampled load is 13.6 job completions per minute according to Figure 41, the total throughput in 39 minutes is 530 jobs (13.6×39) or 40 jobs higher than the CP throughput in 73 minutes.

Furthermore, if the average sampled loading condition used above is replaced with the heaviest sampled user loading condition, the CP throughput changes to 360 job completions (4.95×73), compared to 390 (10×39) for MTS which is a savings of 30 jobs or 8 percent for MTS.

Therefore, at both the average and the maximum sampled user loads, the MTS processes the batch and terminal benchmarks faster than CP plus OS, even when MTS processes the batch and terminal jobs as two sequenced groups.

The Non-EDIT throughput comparisons are also of interest. For example, at the maximum sampled user load, the total Non-EDIT throughput for CP is 110 jobs in 73 minutes (73×1.5 from Figure 9) compared to 117 jobs in 39 minutes for MTS (39×3 from Figure 32) or a difference of 6 percent in the terminal throughputs. However if the load is increased to more than double the average sampled user load (load factors of 95 to 130), the difference becomes much more significant. The total CP Non-EDIT throughput stays relatively constant at 110 to 150 jobs in 73 minutes (Figure 9), but the MTS equivalent increases significantly to 200 to 300 jobs in 39 minutes, which is approximately double that of CP. Therefore, the total batch and terminal Non-EDIT throughputs for the 73 minute period would be about 25

percent higher for MTS than for the split CP and OS system, assuming the improved terminal performance is average over the batch performance.

In all probability, the actual MTS throughputs would probably be somewhere between the 5 percent and 25 percent figures presented above because MTS, with its multiprogramming feature, would probably obtain some overlap of the batch and terminal job processing, and it is really unfair to CP to exclude the EDIT throughputs from the comparisons. In any case, the MTS performance is better than the split CP and OS system.

Section 7

CONCLUSIONS

The following conclusions are made from the results presented in this report. In each case, the reference to the appropriate section or figure is also given.

(1) A basic set of six terminal scripts were developed for loading and benchmarking the performance of different time sharing operating systems (Subsection 2.2.1).

(2) The load factor concept was introduced as a method of calibrating loads. Although the method of selecting the load factor weights was rather arbitrary, the results were good. In most cases the response times, throughputs and effective progress rates were all plotted against the load factors (Section 4.3).

(3) It was verified that the fixed script approach -- whereby each terminal was assigned one and only one script -- is a realistic and accurate technique for terminal benchmarks. The fixed scripts were easier to control, produced results that were easier to analyze and made computer performance easier to compare than mixed scripts, and yet they both produced approximately the same loading effects (Section 5.2 and Table 6).

(4) From the terminal probing tests it was found that the minimum, average, and the maximum sampled user loads corresponded to CP load factor of 25, 50 and 80 respectively. The corresponding MTS load factor were 25, 46 and 74, respectively, and the corresponding combined load factors were

25, 45 and 70 (Tables 7 and 8). Since the benchmarking tests had load factors from 30 to 200, the benchmarking runs represented, light, normal heavy and very heavily overloaded conditions.

(5) CP under average sampled user load provided terminal response times of 0.4, 1.3, 3.4 and 4.5 minutes for FORTRAN, FORTEX, PLISM and PLILG scripts, respectively. These values increased to 1.4, 4.1, 11.1 and 15, respectively, at the maximum sampled user loads (Section 5.4 and Table 7).

(6) Although the CP total effective progress rate was about 3 with the average sampled user load, it was reduced to 0.8 when the load was increased up to the maximum sampled user load. This meant that CP normally operated with a multiprogramming improvement factor of 3, but during overloads CP operated at only 80 percent of the capacity or a serial processing system (Subsection 6.2.4 and Figure 11).

(7) From the above conclusion, it is further concluded that it behooves all designers of time sharing systems to incorporate the load factor concept into their systems and to inhibit initiation of new jobs when the load factor reaches a critical load factor value. (MIS uses this concept in its latest version but CP does not.)

(8) The CP operating system, with 256K bytes of core memory, a single processor and slower disks, out-performed TSS with 768K bytes of memory, two processors and faster disks over almost all tested conditions (Figures 40 and 41 or Section 6.4).

(9) Operating the IBM 360/67 as a split system with OS providing batch services and CP providing terminal services provided better performance than running the full system under TSS, because the batch throughput was obtained for essentially free. (See Conclusion 8 above.)

(10) CP operating with 512K bytes of core memory reduced the response times to about one half and produced twice the throughput as CP with 256K bytes of memory (Figure 22).

(11) MTS out-performed CP under the configurations tested (Section 1.4) by a factor of about three. The MTS terminal response times were approximately 1/3 those of CP (Figure 34 to 36 and 40). The MTS total throughputs were 2 times those of CP (Figures 37 and 41), while the MTS total effective rates were about 10 times those of CP (Figures 38 and 39).

(12) The MTS total effective rates were very impressive ranging from 7.3 to 12.6. The total Non-EDIT effective progress rates were equally impressive ranging from 2 to 3.8 at all loads except the lightest benchmark and the heaviest overload. This meant the MTS operated much better than a series processor (Subsection 6.5.3 and Figure 32).

(13) TSS out-performed OS on batch-only benchmarks by a small margin. This is a conditional conclusion because there is a possibility that the conclusion is the results of the batch benchmark design which was biased against the 768 K byte OS system instead of differences in actual performance. (See discussion in Subsection 6.8.2.)

(14) MTS out-performed TSS and OS on batch-only benchmarks by a wide margin. The MTS total turnaround time was less than half that for OS.

(15) When MTS processed all the jobs in the batch benchmarks and then processed terminal benchmarking jobs, the total throughput for MTS was about five percent better than that processed by CP and OS running separately for the same time periods.

(16) When the total Non-EDIT throughputs were compared, under conditions as specified in Conclusion (15), the MTS total throughputs were seven percent higher CP plus OS totals (Figure 29 for MTS). However, when the Non-EDIT throughputs were compared at higher load factors of 95 to 130, MTS processed nearly twice as many terminal jobs for a total savings of about 25 percent (Subsection 6.8.3).

Based on the above conclusion, it is suggested that MTS provides a superior performance to the split CP-OS system. It is probably that MTS would overlap the execution of terminal and batch job executions to achieve even better performance than described above.

In his thesis, E. F. Hinson has concluded that there are several other advantages besides performance improvements for switching to MTS. In particular the MTS system is a much easier system to learn and to use than the two systems that are currently being used. The ease of learning is an important consideration in any university environment.

Therefore it is strongly recommended that the administration of IBM 360/67 computer centers give serious consideration to the adoption of the Michigan Terminal System for providing both batch and terminal services.

The next major step in evaluating and implementing the MTS system is to confirm that the actual MTS performance is, in fact, as good or better than that suggested above for suitable combinations of batch and terminal loadings. An introduction to the Michigan Terminal System and its features can be obtained from [3] or [16].

REFERENCES

1. "Scheduling TSS/360 for Responsiveness," W. J. Doherty, Proc of the Fall Joint Computer Conference, 1970, pp 97-111.
2. "An Empirical Comparison of the CP/67 and TSS/360 Time Sharing Systems," William R. Haines and James H. Porterfield, NPS, June 1971, MS thesis.
3. "Comparing the Performance of the Michigan Terminal System with CP/67 and TSS/360 on the IBM 360/67 Computer," Elbert F. Hinson, December 1971, MS thesis.
4. "AJOB: A Job Simulator," Pimporn Zeleny, Technical Memorandum, Naval Postgraduate School Computer Center, Monterey, California.
5. "A Remote Terminal Emulator for Loading and Performance Measurement of On-line Systems," D. L. James, Mitre Report M72-83, Summer 72. Also presented at SHARE XXXVIII meeting in San Francisco on March 9, 1972.
6. "Performance Comparison of Two Time-Sharing Systems," M. Fredrich, T. A. Marsland and G. Neufeld, University of Alberta, Edmonton, Alberta, Canada, unpublished report.
7. "Benchmarking Analysis of Time Sharing Systems," A. D. Karush, Systems Development Corporation, Santa Monica, California, April 1969.
8. "Preliminary Steps in Optimizing University Computer Performance Using Hardware and Software Monitors," Robert R. Hanke, NPS, December 1971, MS thesis.
9. "Improving Performance of an IBM 360/67 Computer by Using a Hardware Monitor on the Disk Facility, Jay W. Sprague, NPS, June 1972, MS thesis.
10. "SIPE: A TSS/360 Software Measurement Technique," W. R. Deniston , Proceedings of the 24th National ACM Conference, 1969, p 69.
11. "Improving TSS/360 Performance by Tuning the Table-driven Scheduler," Jerry K. Baird, NPS, June 1971, MS thesis.
12. "MEASURE -- A CP/67 Software Monitor," -- a program obtained from Stuart Wecker, State University of New York, Stony Brook, New York, 1970.
13. "A Simulation Study of the CP/67 Time Sharing System," Brian J. Stanger, University of Alberta, August 1970, MS thesis.

14. "Progress Report on Measurement of Time-Sharing System, E. E. Gregeris and R. F. Belanger, CS 3800 Final Report, Available from G. H. Syms, NPS, June 1971.
15. "Productivity of Multiprogrammed Computers -- Progress in Developing an Analytic Prediction Method," D. J. Lasser, Communications of the ACM, v. 12, No. 12, December 1969, p 679.
16. "MTS -- the Michigan Terminal System, Volume 1: MTS and the Computing Center," Third Edition, University of Michigan Computer Center, Ann Arbor, Michigan, January 1973.
17. "An Emperical Comparison of the CP/67 and TSS/360 Time Sharing Systems," G. H. Syms, W. R. Haines and J. H. Porterfield, Proceedings of the Fifth Hawaii International Conference on System Sciences, January 1972, pp 206-208.

Appendix A

TERMINAL SCRIPTS USED FOR BENCHMARKED COMPARISONS CP/67, TSS/360 AND MTS TIME SHARING SYSTEMS

G. H. SYMS, W. R. HAINES, and J. H. PORTERFIELD, Jr

Naval Postgraduate School
Monterey, California 93940

The purpose of this appendix is to describe in detail and to supply the listing for the scripts used in comparing three time sharing systems for the IBM 360/67. This appendix is to be used in conjunction with a paper entitled "An Empirical Comparison of CP/67 and TSS/360 Time Sharing Systems," published in the Proceeding of the Fifth International Conference on Systems Sciences, 1972, pp 206-208 [17]. Further details on the results of the tests can be obtained from an MS Thesis with the same title by W. R. Haines and J. H. Porterfield, Jr., June 1972, US Naval Postgraduate School [2].

The basic types of programs included in benchmarks are:

- Compilations
- Executions
- Editing

The compilations are represented by 3 programs; a 74 card Fortran compilation called FORTRAN, a small 47 card PL/I compilations called PLISM and a large 434 card PL/I compilation called PLILG. The execution programs are represented by a compute-bound Fortran execution program called FORTEX and a page-bound Fortran execution program called PAGE. PL/I programs were chosen to represent large complex compilers which have a large working sets (30 to 40 pages) and use large amounts of CPU time, thus degrading the performance of any time sharing system. It is believe that these jobs can also represent the load from assembler jobs. The Fortran programs were chosen because they represent the major jobs at the installation under investigation. Each program

was activated by a script. In simple cases the script simply printed the time, called the program and then repeated itself. For most of the tests a terminal executed a single script allowing more rigid control on the test and more accurate analysis. In some cases mixed scripts were used in which a single script on a given terminal was a combination of several simple scripts. As stated in the paper, this provided a means for checking the bias in the fixed-script technique of comparing performance. Before discussing the scripts in detail some of the major evolutionary steps in their development will be presented.

Basically four changes were made to the scripts after the first CP test run:

1. The EDIT script was changed to reflect the think time of the user by activating the EDIT script for 5 seconds out of every 60 seconds of connect time. This was accomplished by hitting the terminal ATTN button after 5 seconds and then restarting the terminal after 55 seconds. If several terminals were editing the starting time were staggered. (For CP only.)
2. The ratio of edit to run (meaning compilation + execution) programs was changed from 5:1 to 2:1, increasing the load by about 2 1/2 times. Also it was felt that 2:1 ratio represented a better estimation of the actual load.
3. The EXEC routines in CP were rewritten to print the desired times at the terminals.
4. An organized plan was formed for combining the scripts on various terminals to represent various loads from medium to heavily overload. The plan was to start with medium load and change the job mix (i.e. benchmark or set of scripts) every 15 minutes with the number of terminals varying from 12 to 22. (Note the load is much more dependent on the run to edit ratio than the number of terminals.)

After run two, it was concluded that the scripts in the present form could be used for a valid comparison of CP and TSS but that further changes in the scripts would make the comparison more realistic; these were:

1. Although the ratio of two edits to one running program seems realistic, it didnot place a heavy enough load on the system. Thus it was decided to vary the ratio of edit to run programs from 1:1 to 1:5 which means that almost all the 24 terminals were compiling or executing programs in the worst case.
2. Since the CP editor had an advantage over the TSS editor by not filing the results after every edit command, it was decided to make the EDIT program function more than the TSS editor (Normally, CP uses an in-core editor and only changes the file on disk when told to do so by the user, but TSS files the results of every edit command on disk. This makes a significant difference in the paging load.) The rational for this change was that most users would probably only make a few changes to their program before filing it and recompiling. This change was accomplished by shortening the EDIT script so that it took about 5 seconds (of real time) to complete and by making it non-looping. Every 60 seconds the EDIT script was initiated again. (This change increased the CP paging load from the EDIT script significantly.)
3. The number of loops in the FORTEX program before an output was produced was decreased from 10^7 to 10^6 in order to increase the amount of output and the accuracy of measuring the load from this script.
4. A program PAGE was added to the benchmark to represent page-bound execution job. The program is a matrix multiplication program that has its matrix elements spread over several pages.

After run three there was one change made to the way the FORTEX script called the program, in order to make it more consistant with the other programs and to print the time at fairly regular intervals. Instead of the program looping, the EXEC program was made to loop and call a non-looping version of FTN. (The self-looping version can be obtained from the one shown later by moving the WRITE statement labelled 1000 to the first statement.)

The final scripts were as follows. (CP examples will be used unless otherwise noted.)

1. EDIT - A CP EXEC routine that performs several edit functions such as locating a string, moving the pointer up and down and typing some output. In CP the routine was non-looping, took about 5 seconds of real time to complete, and was activated every 60 seconds to represent user think time. In TSS a simulated timer was used to wait for 55 seconds before repeating. In MTS the printing of 10 lines was used as an effective delay.
2. FORTTEX - A routine to simulate compute-bound jobs. It executes a loop of 2 additions, a test and a branch 10^6 times and then prints a short line and gives the time.
3. FORTRAN - A routine that compiles an average size (75 card) Fortran program.
4. PLISM - A routine that compiles a small (47 card) PL/I program.
5. PLILG - A routine that compiles a reasonable sized (434 card) PL/I program.
6. PAGE - A routine to represent page-bound jobs. It sequentially accesses a single element from a page of a 30-page matrix.

Table 1 shows the equivalent names for the CP, TSS and MTS scripts and program names. The listing attached include the following actual scripts and programs as run in comparing the time sharing systems:

1. CP EXEC routines for fixed script tests,
2. All the programs used by CP,
3. The CP mixed scripts,
4. The CP mixed script for probing the load under normal operating conditions,
5. The CP EXEC routines used by the mixed scripts,
6. The TSS PROCDEF routines, including the fixed scripts and the mixed scripts. (Note ALL 4 include the PAGE program but there is no equivalent in CP.)
7. MTS files for the fixed scripts (equivalent to CP EXEC routines).

Table 1 EQUIVALENT NAMES FOR SCRIPTS

ALIAS (name in paper)	CP and TSS			MTS	
	EXEC and PROCDEF	Program		Script file	Program
	Looping version	non- looping	Name	(looping)	
EDIT	-	P	LOOP	EDITS	EDITOR
FORTEX	CRUN	CRUNN	FTN *	FORTX	FORTEX
FORTRAN	COMPI	COMPII	LOOP	FORTRAN	FORCOMP
PLISM	COMP2	COMP22	HAI3	PLILIT	PLISM
PLILG	COMP	COMPP	FINK	PLIBC	PLILG
PAGE					
PAGE	PG	PG	PAGE	PAGER	PAGE
Mixed Scripts	ALL1,ALL2, ALL3**	MIX	all above***	--	--

* In actual TSS tests FTN was called FEXEC. Also in latest version of CP tests FTN compiled LOOP instead of the proper program. This version was only used in MIX and not in any of the comparison runs.

** TSS also contained ALL 4 which included PAGE

*** except PAGE, but ALL 4 in TSS did contain PAGE.

ALIAS EDIT

OFFLINE READ P EXEC
 &TYPEOUT ALL TIME
 &BEGSTACK
 L /50/
 N 10
 TOP
 FILE
 &END STACK
 EDIT LOOP FORTRAN
 &TIME

ALIAS FORTEX

OFFLINE READ CRUN EXEC
 &TYPEOUT OFF TIME
 BLIP \$
 LOAD FTN (XEQ)
 &TIME
 &GOTO TOP

OFFLINE READ FTN FORTRAN
 I=0
 10 I=I+1
 A=2. +2.
 IF (I.EQ.1000000) GOTO 1000
 GOTO 10
 1000 WRITE (6,2000)
 2000 FORMAT ('*****')
 END

FTN000
 FTN000
 FTN000
 FTN000
 FTN000
 FTN000
 FTN000
 FTN000

ALIAS FORTRAN

OFFLINE READ COMPI EXEC
 &TYPEOUT ALL TIME
 BLIP \$
 FORTRAN LOOP
 &TIME
 &PRINT **COMPILATION COMPLETE**
 &GOTO TOP

ALIAS PLISM

OFFLINE READ COMP2 EXEC
 &TYPEOUT ALL TIME
 BLIP \$
 PLI HAL3
 &TIME
 &PRINT **COMPILATION COMPLETE**
 &GOTO TOP

ALIAS PLILG

OFFLINE READ COMP EXEC
 &TYPEOUT ALL TIME
 BLIP \$
 PLI FINK
 &TIME
 &PRINT **COMPILATION COMPLETE**
 &GOTO TOP

7	NEERING',/, ' 620 COMMUNICATIONS MANAGEMENT',/, ' 817 NAVAL MANAGEMEP	PAU00540
8	NT',/, ' 999 P.G. SCHOOL STAFF')	PAU00610
	WRITE (6,2)	PAU00560
2	FORMAT ('1',26X,'YEAR CURR PHONE',40X,'NO.OF LAST DUTY')	PAU00570
	WRITE (6,3)	PAU00580
3	FORMAT (' NAME',17X,'RANK GRP NO. NUMBER WIFE ADDRESS',14X	PAU00590
	1,'CITY CHILD STATION',17X,'SMC',/)	
	WRITE (6,4)	PAU00610
4	FORMAT ('+', '-----',17X,'-----'-----'-----',	PAU00620
	114X,'-----',17X,'-----',/)	PAU00630
30	READ (2,8,END=50) XNA,RANK,NYR,NCRS,NTEL,WIFE,ADDR,CITY,NDPN,DUTY	PAU00630
	2,SMC	PAU00650
	WRITE (6,9) XNA,RANK,NYR,NCRS,NTEL,WIFE,ADDR,CITY,NDPN,DUTY,	PAU00650
	1SMC	PAU00670
9	FORMAT (1X,19A1,2X,4A1,2X,I2,3X,I3,1X,I7,1X,9A1,1X,5A4,2X,A1,6X,I	PAU00670
	11,4X,5A2,14X,I4)	PAU00680
	GO TO 30	PAU00700
50	WRITE (6,7) DATE	PAU00710
7	FORMAT (///,40X,5A4)	PAU00720
91	CONTINUE	PAU00730
	STOP	PAU00740
	END	PAU00750

ALIAS PLISM

FFLINE READ	HAI3	PLI	
NEWRAP: PROC OPTIONS (MAIN);			HAI00010
TRY: BEGIN;			HAI00020
DCL I FIXED DEC (3) INITIAL (1);			HAI00030
DCL (RCOT,X,X1,XJ,XI) FLOAT DEC (16) INITIAL (0);			HAI00040
DCL E FIXED DEC (3) INITIAL (0.0001);			HAI00050
DCL BUFFER CHAR (80);			HAI00060
DISPLAY ('INPUT') REPLY (BUFFER);			HAI00070
DISPLAY (BUFFER);			HAI00080
GET STRING (BUFFER) LIST (XJ);			HAI00090
DCL (D,DJ,DI) FLOAT DEC (8) INITIAL (30);			HAI00100
AGAIN: XI=XJ;			HAI00110
IF DF((XI))=0 THEN GOTO TRY;			HAI00120
XJ=XI-F((XI))/DF((XI));			HAI00130
DI=DJ;			HAI00140
DJ=ABS(XJ-XI);			HAI00150
IF DJ>E THEN			HAI00160
STUP: IF DJ>DI THEN			HAI00170
GOTO TRY;			HAI00180
ELSE DO;			HAI00190
IF I>20 THEN			HAI00200
GOTO TRY;			HAI00210
ELSE DO;			HAI00220
DISPLAY ('AT ITERATION ' I ' X' I '=' XJ);			HAI00230
I=I+1;			HAI00240
GOTO AGAIN;			HAI00250
END;			HAI00260
END;			HAI00270
ELSE			HAI00280
IF ABS (F((XJ)))>E THEN			HAI00290
GOTO STUP;			HAI00300
ELSE DO;			HAI00310
DISPLAY ('ROOT = ' XJ);			HAI00320
STOP;END;			HAI00330
DCL F ENTRY (FLOAT DEC (16)) RETURNS (FLOAT DEC (16));			HAI00340
F: PROC (X) FLOAT DEC (16);			HAI00350
DCL X FLOAT DEC (16);			HAI00360
X=X**2-EXP(X)-3;			HAI00370
RETURN (X);			HAI00380
END F;			HAI00390
DCL DF ENTRY (FLOAT DEC (16)) RETURNS (FLOAT DEC (16));			HAI00400
DF: PROC (X) FLOAT DEC (16);			HAI00410
DCL (X,Y) FLOAT DEC (16);			HAI00420
Y=X-1;			HAI00430
X=2*X-X*EXP(Y);			HAI00440
RETURN (X);			HAI00450
END DF;			HAI00460
END NEWRAP;			HAI00470


```

OFFLINE READ
&TYPEOUT OFF
CP Q U
&TIME
& PAGE
&TIME
&GOTO TOP

```

PG

EX EC

```

OFFLINE READ          PAGE          FORTRAN
      DIMENSION A(1030,30)
5      WRITE (6,100)
      DO 10 I=1, 1030
      DO 10 J=1, 30
10     A(I,J)=100.+50.
100    FORMAT(' START OVER ')
      STOP
      END

```

PAG000
PAG000
PAG000
PAG000
PAG000
PAG000
PAG000
PAG000

[illegible]

```

C      PROGRAM TO COMPILE THE ROSTER OF SUBMARINE OFFICERS ATTACHED TO THE NAVAL POST GRADUATE SCHOOL
C      DIMENSION XNA(19),RANK(4),WIFE(9),ADDR(5),DUTY(5),DATE(5)
C      READ THE NUMBER OF DATA CARDS NOT INCLUDING THE YELLOW ONE, THE
C      NUMBER OF COPIES OF THE DIRECTORY DESIRED, AND THE DATE
1      READ(5,1)NCR,NCCP,DATE
C      FORMAT(2I4,5A4)
C      THIS LOOP READS ALL THE DATA CARDS AND PUTS THE DATA ON DISK
      DO 90 I=1,NCR
        READ(5,8)XNA,RANK,NYR,NCRS,NTEL,WIFE,ADDR,CITY,NDPN,DUTY,SMC
        8      FORMAT (19A1,4A1,12,I3,17,9A1,5A4,A1,11,5A2,14)
        90      WRITE (2,8)XNA,RANK,NYR,NCRS,NTEL,WIFE,ADDR,CITY,NDPN,DUTY,SMC
        END FILE 2
C      THIS LOOP PRINTS OUT THE ROSTER, REWINDS THE DISK AND CONTINUES
C      DOING THIS UNTIL NCCP COPIES ARE PRINTED.
      DO 91 I=1,NCCP
        REWIND 2
        WRITE (6,10)
        10      FORMAT ('1. 24X, 'DIRECTORY OF SUBMARINE OFFICERS ATTACHED TO THE NAVAL POSTGRADUATE SCHOOL', ///, ' 1. THIS DIRECTORY IS UNOFFICIAL AND 2D IS INTENDED PRIMARILY FOR SOCIAL REFERENCE.', ///)
        WRITE (6,11)
        11      FORMAT (' 2. THE SUBMARINE LIASON OFFICER IS LCDR PHIL OCONNELL. THIS OFFICE IS LOCATED IN ROOT', /, ' HALL, ROOM 214, ENGINEERING 2SCIENCE CURRICULUM OFFICE. PHONE NUMBER 646 2426.', ///)
        WRITE (6,12)
        12      FORMAT (' 3. THE SUBMARINE OFFICERS SOCIAL CHAIRMAN IS LCDR RAY 1 ANDERSON, PHONE 373 5150.', ///)
        WRITE (6,13)
        13      FORMAT (' 4. THE CHAIRMAN OF THE SUBMARINE WIVES GROUP IS JANE 1 WHITE, PHONE 372 7053.', ///)
        WRITE(6,14)
        14      FORMAT (' 5. DOLPHIN PLAYING CARDS, NAPKINS, CALENDARS, COOKBOOKS, 1ETC, CAN BE OBTAINED OR ORDERED THROUGH', /, ' MRS JOAN EGAN, PHONE 2 375 1710.', ///)
        WRITE (6,15)
        15      FORMAT (' 6. CHANGES OR ADDITIONS TO THIS DIRECTORY SHOULD BE BROUGHT TO THE ATTENTION OF THE SUBMARINE LIASON OFFICER', /, ' OR THE 2 SOCIAL CHAIRMAN.')
        WRITE(6,5)
        5      FORMAT (//////////,1X,'CITY LEGEND:', ///, ' C CARMEL', 11X, ' H CARMEL 1 HEIGHTS V CARMEL VALLEY D DEL REY OAKS M MARINA', /, ' 2P MONTEREY PLEN CC T MONTEREY B PEBBLE BEACH G PACIFI 3C GROVE S SEASIDE', /, ' A SALINAS Q BOQ', ///)
        WRITE (6,6)
        6      FORMAT (' CURRICULUM CODE:', /, ' 360 OPERATIONS ANALYSIS', /, ' 367 MPAU0047 1MANAGEMENT (COMPUTER SYSTEMS)', /, ' 368 COMPUTER SCIENCE', /, ' 372 MEPAU0048 2TEOROLOGY', /, ' 380 ADVANCED SCIENCE', /, ' 440 OCEANOGRAPHY', /, ' 460PAU0049 3 ENGINEERING SCIENCE', /, ' 461 BACHELOR OF ARTS/SCIENCE', /, ' 521 NUPAU0050 4CLEAR ENGINEERING', /, ' 530 WEAPONS ENGINEERING', /, ' 535 UNDER WATEPAU0051 5R PHYSICS', /, ' 570 NAVAL ENGINEERING', /, ' 590 ENGINEERING ELECTRONPAU0052 6ICS', /, ' 600 COMMUNICATIONS ENGINEERING', /, ' 610 AERONAUTICAL ENGP

```

A L I A S P L I L G

```
FLINE READ       FINK       PLI
PROJECT:PROC OPTIONS(MAIN);
```

```
/* THIS PROGRAM WAS DONE BY THE COOPERATIVE EFFORTS OF
J. BAIRD, B. HAINES, R. SPENCER, AND R. WOOLS */
```

```
/* THIS PROGRAM CONSTRUCTS A RING STRUCTURE USING THE
26 LETTERS OF THE ALPHABET AND USES THESE LETTERS
, WHICH CORRESPOND TO THE FIRST LETTER OF A PERSON'S
LAST NAME, AS ENTRY POINTS TO BUILD A SUBRING
STRUCTURE OF RECORDS CONTAINING DATA ITEMS: LAST
NAME, FIRST NAME, OCCUPATION, CITY, AGE.
THE PROGRAM ALLOWS ONE TO INSERT NEW RECORDS,
DELETE RECORDS, CHANGE RECORDS, LOCATE RECORDS
ACCORDING TO LAST NAME AND FIRST NAME, AND PRINT
A LISTING OF ALL RECCRDS IN THE SYSPRINT FILE */
```

```
DCL ALPHABET CHAR(26) INIT('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
DCL ANSWER CHAR (10)VAR ;
DCL BUFFER CHAR(80);
DCL SYSIN FILE STREAM ENVIRONMENT(F(80));
DCL OPEN FILE(SYSIN);
DCL(FIRST_LTR,FRONT,BACK) PTR;
DCL(NLAST,NFIRST,NOCC,NCITY,OLD_LAST,OLD_FIRST) CHAR(10);
DCL (POSITION,LISTING) FIXED BIN(1) INIT(0);
DCL NAGE CHAR(3);
DCL BUILD_ENTRY (CHAR(10));
DCL BUILD_REC_ENTRY (PTR,PTR) ;
DCL LOOK_UP_ENTRY (CHAR(10),CHAR(10)) RETURNS (PTR);
DCL BUILD_ALPHA_ENTRY (PTR) RETURNS (PTR);
DCL FIND_ENTRY (CHAR(10), CHAR(10)) ;
DCL SYSPRINT FILE STREAM PRINT ENV (F(120));
DCL ALPHA_ARRAY (26) PTR;
DCL LABEL LABEL;
DCL ERROR_ENTRY(LABEL);
```

```
DCL 1 ALPHA_BASED (ALPHA_PTR),
2 LTR CHAR(1),
2 NEXT_LTR PTR,
2 LIST_PTR PTR;
```

```
DCL 1 RECORD_BASED(NEW_RECORD),
2 NAME,
3 LAST CHAR(10),
3 FIRST CHAR(10),
2 OCCUPATION CHAR(10),
2 CITY CHAR(10),
2 AGE CHAR(3),
2 NEXT_RECORD PTR;
```

```
/* THIS BLOCK STRUCTURE HANDLES THE TERMINAL I/O FOR THE
PROGRAM */
```

```
ON ENDFILE(SYSIN) BEGIN;
DISPLAY(' ');
DISPLAY(' ');
DISPLAY('DO YOU WANT TO PROCESS ANY RECORDS?');
DISPLAY(' ');
DIS1:DISPLAY('TYPE IN YES OR NO IN QUOTES');
DISPLAY(' ');
DISPLAY(' ') REPLY(BUFFER);
ON ERROR CALL ERROR(DIS1);
GET STRING(BUFFER) LIST(ANSWER);
IF ANSWER='NO' THEN EXIT;
```

```
FIN00010
FIN00020
FIN00030
FIN00040
FIN00050
FIN00060
FIN00070
FIN00080
FIN00090
FIN00100
FIN00110
FIN00120
FIN00130
FIN00140
FIN00150
FIN00160
FIN00170
FIN00180
FIN00190
FIN00200
FIN00210
FIN00220
FIN00230
FIN00240
FIN00250
FIN00260
FIN00270
FIN00280
FIN00290
FIN00300
FIN00310
FIN00320
FIN00330
FIN00340
FIN00350
FIN00360
FIN00370
FIN00380
FIN00390
FIN00400
FIN00410
FIN00420
FIN00430
FIN00440
FIN00450
FIN00460
FIN00470
FIN00480
FIN00490
FIN00500
FIN00510
FIN00520
FIN00530
FIN00540
FIN00550
FIN00560
FIN00570
FIN00580
FIN00590
FIN00600
FIN00610
FIN00620
FIN00630
FIN00640
FIN00650
FIN00660
FIN00670
FIN00680
FIN00690
FIN00700
```

```

IF ANSWER='YES' THEN DO;
    DISPLAY('YOU DIDNT SAY YES OR NO, WHAT THE HELL DO' ||
    ' YOU WANT????');
    GO TO DIS1;
END;
DISPLAY('INDICATE WHAT YOU WANT BY TYPING IN ONE OF THE' ||
    ' FOLLOWING:');

/* THE FOLLOWING GROUPS ARE USED TO INSERT NEW RECORDS,
DELETE RECORDS, AND CHANGE RECORDS WITHIN A SUPRING */

DO WHILE(BUFFER=' ');
    DISPLAY(' ');
    DIS2:DISPLAY('INSERT,DELETE,CHANGE,LOCATE,LISTING OR QUIT' ||
    ' IN QUOTES');
    DISPLAY(' ');
    DISPLAY(' ') REPLY(BUFFER);
    ON ERROR CALL ERROR(DIS2);
    GET STRING(BUFFER) LIST(ANSWER);
    IF ANSWER='INSERT' | ANSWER='DELETE' | ANSWER='CHANGE' |
    ANSWER='LOCATE' | ANSWER='LISTING' | ANSWER='QUIT'
    THEN DO;
    IF ANSWER='INSERT' THEN DO;
        DIS3:DISPLAY('INDICATE WHAT YOU WANT CHANGED BY' ||
        ' TYPING IN ONE OF THE FOLLOWING:');
        DISPLAY('LAST, FIRST, OCCUPATION, CITY, AGE IN QUOTES');
        DISPLAY(' ');
        DISPLAY(' ') REPLY(BUFFER);
        ON ERROR CALL ERROR(DIS3);
        GET STRING (BUFFER) LIST(NLAST,NFIRST,NDOCC,NCITY,
        NAGE);
        CALL BUILD(NLAST);
    END;
    IF ANSWER='DELETE' THEN DO;
        DIS4:DISPLAY('INDICATE WHICH RECORD IS TO BE' ||
        ' DELETED BY TYPING IN THE LASTNAME AND THE FIRST' ||
        ' NAME IN QUOTES')REPLY(BUFFER);
        ON ERROR CALL ERROR(DIS4);
        GET STRING(BUFFER) LIST(NLAST,NFIRST);
        CALL FIND(NLAST,NFIRST);
        IF POSITION=0 THEN
            IF FRONT->NEXT_RECORD=NULL & BACK= ALPHA_PTR
            THEN DO;
                LIST_PTR=NULL;
                FREE_RECORD;
            END;
            ELSE DO;
                BACK->NEXT_RECORD=FRONT->NEXT_RECORD;
                FREE_RECORD;
            END;
        END;
    IF ANSWER='LISTING' THEN DO;
        LISTING=1;
        CALL PRINT;
        LISTING=0;
    END;
    IF ANSWER='LOCATE' THEN DO;
        DIS10:DISPLAY('INDICATE WHOSE RECORD YOU' ||
        ' WANT LOCATED BY TYPING IN THE LAST AND ' ||
        ' FIRST NAME IN QUOTES')REPLY(BUFFER);
        ON ERROR CALL ERROR(DIS10);
        GET STRING(BUFFER) LIST(NLAST,NFIRST);
        CALL FIND(NLAST,NFIRST);
        IF POSITION=0 THEN DISPLAY(LAST || FIRST || OCCUPATION ||
        CITY || AGE);
    END;
    IF ANSWER='CHANGE' THEN DO;
        DIS5:DISPLAY('CHANGE WHAT?');
        DISPLAY('INDICATE EITHER NAME, CITY, ' ||
        ' OCCUPATION, OR AGE IN QUOTES')REPLY(BUFFER);
        ON ERROR CALL ERROR(DIS5);
        GET STRING (BUFFER) LIST(ANSWER);

```

```

FIN00710
FIN00720
FIN00730
FIN00740
FIN00750
FIN00760
FIN00770
FIN00780
FIN00790
FIN00800
FIN00810
FIN00820
FIN00830
FIN00840
FIN00850
FIN00860
FIN00870
FIN00880
FIN00890
FIN00900
FIN00910
FIN00920
FIN00930
FIN00940
FIN00950
FIN00960
FIN00970
FIN00980
FIN00990
FIN01000
FIN01010
FIN01020
FIN01030
FIN01040
FIN01050
FIN01060
FIN01070
FIN01080
FIN01090
FIN01100
FIN01110
FIN01120
FIN01130
FIN01140
FIN01150
FIN01160
FIN01170
FIN01180
FIN01190
FIN01200
FIN01210
FIN01220
FIN01230
FIN01240
FIN01250
FIN01260
FIN01270
FIN01280
FIN01290
FIN01300
FIN01310
FIN01320
FIN01330
FIN01340
FIN01350
FIN01360
FIN01370
FIN01380
FIN01390
FIN01400
FIN01410
FIN01420
FIN01430
FIN01440
FIN01450

```


IF ANSWER='NAME' THEN DO;	FIN01460
DIS6:DISPLAY('TYPE IN THE FOLLOWING ITEMS IN '	FIN01470
'QUOTES :OLD_LAST,OLD_FIRST,NEW_LAST,NEW_FI'	FIN01480
'RST')REPLY (BUFFER);	FIN01490
ON ERROR CALL ERROR(DIS6);	FIN01500
GET STRING(BUFFER) LIST(OLD_LAST,OLD_FIRST,	FIN01510
NLAST,NFIRST);	FIN01520
CALL FIND(OLD_LAST,OLD_FIRST);	FIN01530
IF POSITION=0 THEN DO;	FIN01540
BACK->NEXT_RECIRD=FRONT->NEXT_RECORD;	FIN01550
NOCC=OCCUPATION;	FIN01560
NCITY=CITY;	FIN01570
NAGE=AGE;	FIN01580
FREE RECORD;	FIN01590
CALL BUILD(NLAST);	FIN01600
END;	FIN01610
END;	FIN01620
IF ANSWER='CITY' THEN DO;	FIN01630
DIS7:DISPLAY('TYPE IN LAST, FIRST AND THE NEW CITY'	FIN01640
'IN QUOTES')REPLY(BUFFER);	FIN01650
ON ERROR CALL ERROR(DIS7);	FIN01660
GET STRING(BUFFER) LIST(OLD_LAST,OLD_FIRST,NCITY	FIN01670
);	FIN01680
CALL FIND(OLD_LAST,OLD_FIRST);	FIN01690
IF POSITION=0 THEN CITY=NCITY;	FIN01700
END;	FIN01710
IF ANSWER='OCCUPATION' THEN DO;	FIN01720
DIS8:DISPLAY('TYPE IN LAST,FIRST AND NEW'	FIN01730
'OCCUPATION IN QUOTES')REPLY (BUFFER);	FIN01740
ON ERROR CALL ERROR(DIS8);	FIN01750
GET STRING(BUFFER) LIST(OLD_LAST,OLD_FIRST,	FIN01760
NOCC);	FIN01770
CALL FIND (OLD_LAST,OLD_FIRST);	FIN01780
IF POSITION=0 THEN OCCUPATION=NOCC;	FIN01790
END;	FIN01800
IF ANSWER='AGE' THEN DO;	FIN01810
DIS9:DISPLAY('TYPE IN LAST,FIRST AND NEW AGE'	FIN01820
'IN QUOTES')REPLY(BUFFER);	FIN01830
ON ERROR CALL ERROR(DIS9);	FIN01840
GET STRING(BUFFER) LIST(OLD_LAST,OLD_FIRST,NAGE)	FIN01850
;	FIN01860
CALL FIND(OLD_LAST,OLD_FIRST);	FIN01870
IF POSITION=0 THEN AGE=NAGE;	FIN01880
END;	FIN01890
IF ANSWER='NAME' ANSWER='CITY' ANSWER='AGE'	FIN01900
ANSWER='OCCUPATION' THEN DO;	FIN01910
DISPLAY('THE INDICATED RECORD HAS BEEN CHANGED '	FIN01920
'TO READ:');	FIN01930
DISPLAY(LAST FIRST OCCUPATION CITY AGE);	FIN01940
END;	FIN01950
ELSE DO;	FIN01960
DISPLAY('I CANT FIGURE OUT WHAT YOU WANT '	FIN01970
'CHANGED; I CAN ONLY ASSUME THAT:');	FIN01980
CALL ERROR(DIS5);	FIN01990
END;	FIN02000
END;	FIN02010
IF ANSWER='QUIT' THEN DO;	FIN02020
DISPLAY('YOU HAVE INDICATED THAT NO MORE WORK IS TO '	FIN02030
'BE DONE; THEREFORE I WILL PUT A CURRENT LIST'	FIN02040
'ING IN SYSPRINT BEFORE EXITING:');	FIN02050
CALL PRINT;	FIN02060
EXIT;	FIN02070
END;	FIN02080
END;	FIN02090
ELSE DO;	FIN02100
DISPLAY('I CANT FIGURE OUT WHAT IT IS YOU WANT DONE'	FIN02110
'; ICAN ONLY ASSUME THAT:');	FIN02120
CALL ERROR(DIS2);	FIN02130
END;	FIN02140
ND;	FIN02150
ND;	FIN02160
	FIN02170
	FIN02180
	FIN02190
	FIN02200

THIS GROUP OF 8 STATEMENTS IS THE MAIN PROGRAM WHICH OBTAINS THE DATA FROM THE SYSIN FILE AND CALLS THE

```
BUILD PROCEDURE TO BUILD A SUBRING STRUCTURE (DATA
RECORDS) */
```

```
DO I=1 TO 26;
  ALPHA_ARRAY(I)=NULL;
END;
FIRST_LTR=NULL;
DO WHILE('1'B);
  GET LIST(NLAST,NFIRST,N OCC,N CITY,N AGE);
  CALL BUILD (NLAST);
END;
```

```
/* THE FOLLOWING PROCEDURE ALLOWS FOR CORRECTION OF TERMINAL
INPUT ERRORS */
```

```
ERROR:PROC (LABEL);
  DCL LABEL LABEL;
  DISPLAY(' ');
  DISPLAY('THERE IS AN ERROR IN THE INPUT');
  DISPLAY('THE LAST INPUT FROM THE TERMINAL WAS:');
  DISPLAY(' ');
  DISPLAY(BUFFER);
  DISPLAY(' ');
  DISPLAY('CHECK IT THEN RE-INPUT THE CORRECT DATA AS FOLLOWS:');
  GO TO LABEL;
END ERROR;
```

```
/* THIS FUNCTION CALLS PROCEDURES WHICH CONSTRUCT (1) A
RING WHICH HAS THE FIRST LETTER OF A PERSON'S LAST NAME
AND (2) A SUBRING WHICH HAS DATA ITEMS: LAST NAME, FIRST
NAME, OCCUPATION,CITY,AGE */
```

```
BUILD: PROC(NLAST) ;
  DCL NLAST CHAR(10);
  IF FIRST_LTR=NULL THEN DO;
    ALPHA_PTR=BUILD_ALPHA(FIRST_LTR);
    CALL BUILD_REC (ALPHA_PTR,FRONT);
    RETURN;
  END;
  ELSE DO;
    ALPHA_PTR=FIRST_LTR;
    DO WHILE('1'B);
      IF SUBSTR(NLAST,1,1)~=ALPHA_PTR->LTR THEN DO;
        ALPHA_PTR=ALPHA_PTR->NEXT_LTR;
        IF ALPHA_PTR=FIRST_LTR THEN DO;
          ALPHA_PTR=BUILD_ALPHA(FIRST_LTR);
          CALL BUILD_REC (ALPHA_PTR,FRONT);
          RETURN;
        END;
      END;
    END;
    ELSE DO;
      FRONT=LOOK_UP (NLAST,NFIRST);
      IF POSITION~=0 THEN
        CALL BUILD_REC (ALPHA_PTR,FRONT);
      RETURN;
    END;
  END;
  RETURN;
END BUILD;
```

```
/* THE FOLLOWING PROCEDURE BUILDS A SUBRING, UNDER THE FIRST
LETTER OF A PERSON'S LAST NAME, WHICH CONTAINS DATA ITEMS:
LAST NAME, FIRST NAME, OCCUPATION, CITY, AGE */
```

```
BUILD_REC: PROC (ALPHA_PTR,FRONT);
  DCL (ALPHA_PTR,FRONT) PTR;
  ALLOCATE RECORD;
  LAST=NLAST;
  FIRST=NFIRST;
```

```

OCCUPATION=NOCC;
CITY=NCITY;
AGE=NAGE;
IF ALPHA_PTR=FRONT THEN
  IF LIST_PTR=NULL THEN DO;
    NEXT_RECORD=NULL;
    LIST_PTR=NEW_RECORD;
    RETURN;
  END;
ELSE DO;
  NEXT_RECORD=LIST_PTR;
  LIST_PTR=NEW_RECORD;
  RETURN;
END;
ELSE DO;
  NEXT_RECORD=FRONT->NEXT_RECORD;
  FRONT->NEXT_RECORD=NEW_RECORD;
  RETURN;
END;
END BUILD_REC;

```

```

/* THE FOLLOWING PROCEDURE LOOKS TO SEE WHERE DATA ITEMS, LAST
LAST AND FIRST NAME OF A RECORD, ARE TO BE INSERTED IN THE
SUBRING */

```

```

LOOK_UP:  PRCC (NLAST,NFIRST) PTR;
          DCL (NLAST,NFIRST) CHAR(10);
          FRONT=ALPHA_PTR->LIST_PTR;
          BACK=ALPHA_PTR;
          POSITION =1;
          DO WHILE (FRONT->NEXT_RECORD=NULL);
            IF NLAST||NFIRST>FRONT->LAST||FRONT->FIRST THEN DO;
              BACK=FRONT;
              FRONT=FRONT->NEXT_RECORD;
            END;
            ELSE IF NLAST||NFIRST=FRONT->LAST||FRONT->FIRST
              THEN DO;
                POSITION=0;
                RETURN(FRONT);
              END;
            ELSE RETURN(BACK);
          END;
          RETURN(BACK);
END LOOK_UP;

```

```

/* THE FOLLOWING PROCEDURE BUILDS THE RING WHICH CONTAINS THE
LETTERS OF THE ALPHABET. THE LETTERS OF THIS RING ARE THEN
USED AS ENTRY POINTS FOR THE INSERTION OF RECORDS CONTAINING
THE FIRST LETTER OF A PERSON'S LAST NAME CORRESPONDING
TO THE LETTER OF THE ALPHABET IN THE RING */

```

```

BUILD_ALPHA:  PRCC (FIRST_LTR) PTR;
              DCL FIRST_LTR PTR;
              DCL TEMP2 FIXED BIN(15);
              ALLOCATE ALPHA;
              TEMP2=INDEX(ALPHABET,SUBSTR(NLAST,1,1));
              ALPHA_ARRAY(TEMP2)=ALPHA_PTR;
              LTR=SUBSTR(NLAST,1,1);
              LIST_PTR=NULL;
              FRONT=ALPHA_PTR;
              IF FIRST_LTR=NULL THEN DO;
                FIRST_LTR=ALPHA_PTR;
                NEXT_LTR=FIRST_LTR;
              END;
              ELSE DO;

```

FIN02970
 FIN02980
 FIN02990
 FIN03000
 FIN03010
 FIN03020
 FIN03030
 FIN03040
 FIN03050
 FIN03060
 FIN03070
 FIN03080
 FIN03090
 FIN03100
 FIN03110
 FIN03120
 FIN03130
 FIN03140
 FIN03150
 FIN03160
 FIN03170
 FIN03180
 FIN03190
 FIN03200
 FIN03210
 FIN03220
 FIN03230
 FIN03240
 FIN03250
 FIN03260
 FIN03270
 FIN03280
 FIN03290
 FIN03300
 FIN03310
 FIN03320
 FIN03330
 FIN03340
 FIN03350
 FIN03360
 FIN03370
 FIN03380
 FIN03390
 FIN03400
 FIN03410
 FIN03420
 FIN03430
 FIN03440
 FIN03450
 FIN03460
 FIN03470
 FIN03480
 FIN03490
 FIN03500
 FIN03510
 FIN03520
 FIN03530
 FIN03540
 FIN03550
 FIN03560
 FIN03570
 FIN03580
 FIN03590
 FIN03600
 FIN03610
 FIN03620
 FIN03630
 FIN03640
 FIN03650

```

        NEXT_LTR=FIRST_LTR->NEXT_LTR;
        FIRST_LTR->NEXT_LTR=ALPHA_PTR;
    END;
    RETURN (ALPHA_PTR);
END BUILD_ALPHA;

```

```

/* THE FOLLOWING PROCEDURE PRINTS OUT A LISTING OF THE ENTIRE
   STRUCTURE OR A LISTING OF THE SYSPRINT FILE */

```

```

PRINT: PROC;
    DCL (TEMP) PTR;
    IF LISTING=0 THEN OPEN FILE (SYSPRINT);
    DO I=1 TO 26;
        IF ALPHA_ARRAY(I)~=NULL THEN DO;
            TEMP=ALPHA_ARRAY(I);
            NEW_RECORD=TEMP->LIST_PTR;
            DO WHILE(NEW_RECORD~=NULL);
                IF LISTING=0 THEN PUT EDIT(LAST, FIRST, OCCUPATION,
                    CITY, AGE)((SKIP, A(10), A(10), A(10), A(10), A(3)));
                ELSE DISPLAY(LAST||FIRST||OCCUPATION||CITY||AGE);
                NEW_RECORD=NEW_RECORD->NEXT_RECORD;
            END;
        END;
    END;
    RETURN;
END PRINT;

```

```

/* THE FOLLOWING PROCEDURE LOCATES THE LETTER OF THE ALPHABET
   CORRESPONDING TO THE FIRST LETTER OF A PERSON'S LAST
   NAME AND SEARCHES THROUGH THE SUBRING FOR THE LAST
   NAME AND FIRST NAME OF THE RECORD DESIRED */

```

```

FIND: PROC (NLAST, NFIRST);
    DCL (NLAST, NFIRST) CHAR(10);
    ALPHA_PTR=FIRST_LTR;
    IF FIRST_LTR=NEXT_LTR THEN IF SUBSTR(NLAST, 1, 1)~=LTR
        THEN DISPLAY('RECORD NOT IN FILE, YOU BLEW IT ');
    ELSE DO;
        NEW_RECORD=LOOK_UP(NLAST, NFIRST);
        IF POSITION~=0 THEN DISPLAY('RECORD NOT IN FILE, '||
            'YOU BLEW IT');
        RETURN;
    END;
    ELSE IF SUBSTR(NLAST, 1, 1)=LTR THEN DO;
        NEW_RECORD=LOOK_UP(NLAST, NFIRST);
        IF POSITION~=0 THEN DISPLAY('RECORD NOT IN FILE, '||
            'YOU BLEW IT');
        RETURN;
    END;
    ELSE DO;
        ALPHA_PTR=ALPHA_PTR->NEXT_LTR;
        DO WHILE(ALPHA_PTR~=FIRST_LTR);
            IF SUBSTR(NLAST, 1, 1)=LTR THEN DO;
                NEW_RECORD=LOOK_UP(NLAST, NFIRST);
                IF POSITION~=0 THEN DISPLAY('RECORD NOT '||
                    'IN FILE, YOU BLEW IT');
                RETURN;
            END;
            ELSE ALPHA_PTR=ALPHA_PTR->NEXT_LTR;
        END;
        IF ALPHA_PTR=FIRST_LTR THEN
            DISPLAY('RECORD NOT IN FILE, YOU BLEW IT ');
    END;
END FIND;
END PROJECT;

```


CP MIXED SCRIPT 1

```

OFFLINE READ      ALL1      EXEC
&PRINT ALL1
&TPYEOUT ALL TIME
CP Q U
EXEC P
EXEC COMPP
EXEC P
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC COMPII
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC P
EXEC P
EXEC COMP22
CP Q U
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC P
EXEC CRUNN
EXEC CRUNN
CP Q N
&TIME
&PRINT END CYCLE
&GOTO TOP

```

CP MIXED SCRIPT 2

```

OFFLINE READ      ALL2      EXEC
&PRINT ALL2
&TPYEOUT ALL TIME
CP Q U
EXEC CRUNN
EXEC CRUNN
EXEC P
EXEC COMP22
CP Q U
EXEC P
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC P
EXEC COMPII
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC P
EXEC P
EXEC COMPP
CP Q U
EXEC P
EXEC P
EXEC P
EXEC CRUNN
EXEC CRUNN
CP Q N
&TIME
&PRINT END CYCLE
&GOTO TOP

```

CP MIXED SCRIPT 3

OFFLINE READ ALL3 EXEC
&PRINT ALL3
&TPYEOUT ALL TIME
CP Q U
EXEC P
EXEC P
EXEC CRUNN
CP Q U
EXEC COMPI I
EXEC P
EXEC P
EXEC CRUNN
EXEC CRUNN
CP Q U
EXEC P
EXEC COMPP
CP Q U
EXEC P
EXEC P
EXEC P
EXEC P
EXEC CRUNN
EXEC P
CP Q U
EXEC CRUNN
EXEC COMP22
EXEC P
EXEC P
EXEC P
EXEC CRUNN
CP Q N
&TIME
&PRINT END CYCLE
&GOTO TOP

CP MIXED SCRIPT FOR MEASURING LOAD UNDER NORMAL OPERATING CONDITIONS

CP67USERID 0770 06/08/71 14.20.27 *****

```

OFFLINE READ      MIX      EXEC
&TYPEOUT OFF
&PRINT >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< *
&PRINT >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< ***
&PRINT START MIX BENCHMARK FOR PROBING SYSTEM LOAD
BLIP
&TIME
&PRINT =====
CP Q U
CP Q N
&PRINT =====
EXEC P
&PRINT =====
EXEC COMPP
&PRINT =====
EXEC CRUNN
&PRINT =====
CP Q U
CP Q N
&PRINT =====
EXEC COMPII
&PRINT =====
EXEC PG
&PRINT =====
EXEC COMP22
&PRINT =====
EXEC P
&PRINT =====
CP Q U
CP Q N
&PRINT =====
&TIME
&PRINT END MIX BENCHMARK
&PRINT >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< ***
&PRINT >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< >>>>>>> <<<<<<<< ***
&PRINT
&PRINT
&PRINT THIS TERMINAL IS NOW AVAILABLE FOR YOUR USE...
&PRINT
&PRINT PLEASE TEAR OFF COPY AND LEAVE IT NEXT TO TERMINAL...
&PRINT TERMINAL WILL LOG ITSELF OFF.....THANK YOU.
&PRINT
&PRINT
&PRINT
CP LOG
    
```

EXEC ROUTINES FOR MIXED SCRIPTS

LAST LETTER DUPLICATED INDICATES NO LOOPING, OTHERWISE
ROUTINE IS THE SAME, EXCEPT FOR PRINT OUT AND TIME DISPLAY

A L I A S E D I T

OFFLINE READ P EXEC
&TYPEOUT ALL TIME
&BEGSTACK
L 750/
N 10
TOP
FILE
&END STACK
EDIT LOOP FORTRAN
&TIME

A L I A S F O R T E X

OFFLINE READ CRUNN EXEC
&TYPEOUT OFF TIME
&PRINT START CRUNN *****
&TIME
LOAD FTN (XEQ)
&TIME
&PRINT CRUNN COMPLETE *****

A L I A S F O R T R A N

OFFLINE READ COMPII EXEC
&TYPEOUT OFF
&PRINT START COMPII*****
&TIME
FORTRAN LOOP
&TIME
&PRINT **COMPI COMPLETE**

A L I A S P L I S M

OFFLINE READ COMP22 EXEC
&TYPEOUT OFF
&PRINT START COMP2*****
&TIME
PLI HAI3
&TIME
&PRINT **COMP2 COMPLETE**

A L I A S P L I L G

OFFLINE READ CUMPP EXEC
&TYPEOUT OFF
&PRINT START CUMPP*****
&TIME
PLI FINK
&TIME
&PRINT **COMP COMPLETE**

A L I A S P A G E

OFFLINE READ PG EXEC
&TYPEOUT OFF
&PRINT START PAGING*****
&TIME
LOAD PAGE (XEQ)
&TIME
&PRINT PAGING COMPLETE*****

P67USERID 0770
FFLINE READ TSS PLI
LOGON GREGERIS

PROCDEF P

DISPLAY 'START TSS EDIT'
CLOCK
DEFAULT SYSINX=E
EDIT SOURCE,LOOP
LOCATE 0, LAST, STRING='50'
LIST 310
LIST 100
END
DEFAULT SYSINX=G
CLOCK
DISPLAY 'END EDIT'
_END

PROCDEF CRUNN

DISPLAY 'START CRUNN'
CLOCK
CALL FTN
CLOCK
DISPLAY 'END CRUNN'
_END

PROCDEF COMPII

DISPLAY 'START COMPII'
CLOCK
ERASE USERLIB (LOOP)
FTN LOOP,Y
CLOCK
DISPLAY 'END COMPII'
_END

PROCDEF COMP22

DISPLAY 'START COMP22'
CLOCK
PLI HA13,Y
CLOCK
DISPLAY 'END COMP22'
_END

PROCDEF COMPP

DISPLAY 'START COMPP'
CLOCK
PLI FINK,Y
CLOCK
DISPLAY 'END COMPP'
_END

PROCDEF PG

DISPLAY 'START PAGING'
CLOCK
CALL PAGE
CLOCK
DISPLAY 'END PAGING'
_END

PR0001650
PR0001670
PR0001680
PR0001690
PR0001700
PR0001710
PR0001720
PR0001730
PR0001740
PR0001750
PR0001760
PR0001770
PR0001780

PR000110
PR000120
PR000130
PR000140
PR000150
PR000160
PR000170

PR000620
PR000630
PR000640
PR000650
PR000660
PR000670
PR000680
PR000690

PR000040
PR000050
PR000060
PR000070
PR000080
PR000090

PR000540
PR000550
PR000560
PR000570
PR000580
PR000590
PR000600

PR001430
PR001440
PR001450
PR001460
PR001470
PR001480
PR001490

ROCDEF ALL3	PR001080
ISPLAY 'START ALL3'	PR001090
LOCK	PR001100
ID 5	PR001110
	PR001120
	PR001130
	PR001140
RUNN	PR001150
ID 5	PR001160
OMPII	PR001170
	PR001180
	PR001190
RUNN	PR001195
RUNN	PR001200
ID 5	PR001210
	PR001220
UMPP	PR001230
ID 5	PR001240
	PR001250
	PR001260
	PR001270
	PR001280
RUNN	PR001290
	PR001300
ID 5	PR001310
RUNN	PR001320
OMP22	PR001330
	PR001340
	PR001350
	PR001360
RUNN	PR001370
ID 1	PR001380
LOCK	PR001390
ISPLAY 'END CYCLE'	PR001400
LL3	PR001410
END	
ROCDEF MIX	PR001510
ISPLAY 'START STAND ALONE MIX'	PR001520
LOCK	PR001530
ID 5	PR001540
ID 1	PR001550
	PR001560
UMPP	PR001570
RUNN	PR001580
OMPII	PR001590
OMP22	PR001600
G	PR001610
LOCK	PR001620
ISPLAY 'END MIX'	PR001630
END	PR001640

MTS SCRIPTS

EDIT SCRIPT

```
$$RUN *TIME
$$RUN *ED;SCARDS=EAS:EDITS
$$RUN *TIME
$CONTINUE WITH EAS:EDITOR
      FILE EAS:EDITS CONTAINS
XEC $EDT
EDIT EAS:LOOP
TCP: SCAN@A *F 26 '50'
L *F
+3
L *L
L *F
P 10 20
STOP
$EDT
```

FORTRAN SCRIPT

```
$$RUN *TIME
$$RUN *FORTRAN;SCARDS=EAS:FORTRAN PAR=NOSOURCE,NOMAP,NOLOAD
$$RUN *TIME
$$CONTINUE WITH EAS:FORCOMP
```

FORTEX SCRIPT

```
$$RUN *TIME
$$RUN EAS:FORTX
$$RUN *TIME
$$CONTINUE WITH EAS:FORTX
```

PAGE SCRIPT

```
$$RUN *TIME
$$RUN EAS:PAGE
$$RUN *TIME
$$CONTINUE WITH EAS:PAGER
```

PLISM SCRIPT

```
$$RUN *TIME
$$RUN *PL1;SCARDS=EAS:PLISM PAR=NS,NLD,NS2,NOL
$$RUN *TIME
$$CONTINUE WITH EAS:PLILIT
```

PLILG SCRIPT

```
$$RUN *TIME
$$RUN *PL1;SCARDS=EAS:PLILG PAR=NS,NLD,NS2,NOL
$$RUN *TIME
$$CONTINUE WITH EAS:PLIBG
```

Appendix B

BATCH BENCHMARKING PROGRAMS

Seventeen FORTRAN and assembler jobs were collected as a batch benchmarking jobstream. The jobs contained one to seven subroutines, required region sizes of 58K bytes to 300K bytes, produced one to fourteen pages of output and required 5 seconds to 360 seconds of CPU time. A summary of the jobs in the batchmark is shown in Tables B1 and B2. A summary of the CPU times required is given in Table B3.

The benchmark programs were arranged in three benchmarks called Jobstreams A, B and C. Jobstreams A and B contained 15 programs SYS001 through SYS015, whereas Jobstream C contained two extra programs, SYS13A and SYS14A. Jobstream A and B included compile, link and go steps with program listings and no listings, respectively. In both cases, the writer was started, then the reader and initiator were started together. For Jobstream C, the programs were precompiled on a disk so that only execution steps were performed. All the JCL (Job Control Language) and data decks were read before the initiation was started.

The most significant data collected was the time from the start of the card reader (or the initiator in the case of Jobstream C) until the last line was printed. In some cases, the time from the start of the third job being initiated until the time for the third-last job finishing was also recorded. Also, the time in which at least two jobs were multi-programming was also recorded.

A complete listing of the batch benchmarking programs can be obtained by writing to the author or a copy of the jobstreams can be obtained by sending a blank magnetic tape.

Table B1: Batch Benchmarking Jobstream Characteristics

Job Name	No. of subroutines		Execution		Input data	Total CPU ~ sec.
	FORTRAN	ASSEMBLER	Region-k	pages of output		
SYS001	7	-	58	1	No	10
SYS002	6	-	100	14	Yes	12
SYS003	2	-	58	5	No	2
SYS004	8	-	82	5	Yes	34
SYS005	5	-	58	2	Yes	6
SYS006	1	1	58	12	No	7
SYS007	4	-	110	4	Yes	9
SYS008	3	1	58	2	No	6
SYS009	7	-	58	9	Yes	13
SYS010	2	-	58	8	No	5
SYS011	2	-	58	1	No	6
SYS012	1	1	58	12	No	7
SYS013	3	-	140	3	Yes	359
SYS014	1	-	300	1	No	266
SYS015	1	-	300	1	No	266
SYS13A	3	-	140	3	Yes	359
SYS14A	1	-	300	1	No	266

Table B2: Jobstream Summary by Region and CPU Time

Jobstream	No. of jobs for region size						No. of jobs with total CPU time ~ sec							
	58K	83K	100K	110K	140K	300K	≤2	≤5	≤10	≤15	≤30	≤60	≤300	≤360
A	9	1	1	1	1	2	1	1	7	2	-	1	2	1
B	9	1	1	1	1	2	1	1	7	2	-	1	2	1
C	9	1	1	1	2	3	4	6	1	-	1	-	3	2

Table B3: Jobstream C:Execution and Region Used

Job Name	Region k	OS/MVT CPU Time (sec.)
SYS001	58	6.63
SYS002	100	4.29
SYS003	58	0.42
SYS004	82	19.51
SYS005	58	0.75
SYS006	58	2.56
SYS007	110	2.07
SYS008	58	0.30
SYS009	58	4.13
SYS010	58	2.20
SYS011	58	0.11
SYS012	58	2.53
SYS013	140	342.94
SYS13A	140	345.00
SYS014	300	261.46
SYS14A	300	257.82
SYS015	300	255.44

Appendix C

CP INTERMEDIATE TERMINAL RESULTS

This appendix presents the intermediate results from the CP terminal tests that were used to prepare the graphs in Section 6, especially Subsection 6.2. These intermediate results are presented here for easy reference or further analysis that might be desirable.

Table C1 presents the average response time at each terminal for the CP tests. This table was prepared by averaging all response times that were obtained from the terminal printout for each individual terminal. For example, for a benchmark test running four EDIT scripts, there should be four mean terminal response times. Runs R21 to R25 are an exception since they present only the mean response time for a script (averaged overall terminals running that script).

Table C2 presents the CP mean response times for each script which were obtained by averaging the response times at all terminals running the same script. The mean response time was used in determining the calculated throughput as shown in Table C3.

Tables C4 and C5 show the terminal and script measured throughputs, respectively. The measured throughput was determined by counting the number of job completions and dividing the run duration.

Table C6 presents the minimum, or stand-alone, CP response times for each script as determined by one of several methods. The minimum response

times were used in calculating the effective progress rate as shown in Table C7. The total effective progress rate is particularly interesting since it provides a direct comparison with a serial processing computer, as discussed in Subsection 6.2.4.

Table C8 presents two of the software monitor results for all CP runs, namely, the percentage of CPU time in the problem state and the page reading rates. Table C9 presents a more detailed version of the results from the software monitor for Test 3 (Runs R31 to R37).

Table C10 presents the individual terminal response times that were used to compare the CP performances when operating with one, two and three core boxes. Unfortunately the run with three core boxes was so short that it is considered unreliable and was not included in the analysis and results presented in Section 6. The results for the runs with one and two core boxes are used to compare the CP performance with TSS which had three core boxes.

Table C1 CP - Average Response Time at Each Terminal (in minutes:seconds)

RUN NUMBER	CP LOAD FACTOR	RESPONSE TIME						NO. EDIT	NO. PAGE	NO. FORTEX	NO. PLISM	NO. PLILG	NO. FORTRAN
		EDIT	PAGE	FORTEX	PLISM	PLILG	FORTRAN						
21	32							8	0	3	0	0	1
22	32							10	0	2	0	0	1
23	48					4:48		12	0	3	0	1	1
24	60					6:24		14	0	3	0	1	3
25	64					5:36		15	0	4	1	1	1
R31	110	1:38 1:00 :37 :26			8:29 8:21 8:44	22:31 22:54 22:59 21:43	2:46 2:45 2:44 2:47 2:40 2:39 2:45	4	0	6	3	4	7
R32	86	:49 1:05 :56 1:15 :52 :45 1:23 1:34 2:40 2:14 2:04 1:34			10:22 10:24 8:35	25:08 31:36	2:52 2:46 3:08 2:48 2:55	12	0	2	3	2	5
R33	130	:49 1:02 :50 1:15 1:03 :44 1:26 1:43 2:06 2:22			12:48 14:14 14:07	31:06 25:36	4:30 4:42 4:48 5:03 4:40	10	2	2	3	2	5

Table C1 (continued)

RUN NUMBER	CP LOAD FACTOR	RESPONSE TIME						NO. EDIT	NO. PAGE	NO. FORTEX	NO. PLISM	NO. PLILG	NO. FORTRAN
		EDIT	PAGE	FORTEX	PLISM	PLILG	FORTRAN						
R34	174	:45 :29 :51 :47 :38 :34 1:07 1:16			14:01 15:09 14:04	41:58 39:11	5:01 5:20 5:08 5:10 5:11	8	4	2	3	2	5
R35	218	:49 :43 :42 1:10 :55 :47			15:10 14:22 16:23		3:17 8:31 7:06 3:51 4:56	6	6	2	3	2	5
R41	66	1:02 1:02 1:06 :44 :54 1:00 :46 :52 1:09 :58 :40 :48 :51 :45		5:33 5:41 4:56 6:01 6:19	3:12	12:31	1:29	14	0	5	1	1	1
R42	68	1:00 :51 :57 1:00 :55 :50 :52 1:06 1:09 1:07 :57 :48 :41		5:12 5:04 5:08 5:07 4:56 5:06	3:38	11:26	1:16	13	0	6	1	1	1

Table C1 (continued)

RUN NUMBER	CP LOAD FACTOR	RESPONSE TIME						NO. EDIT	NO. PAGE	NO. FORTEX	NO. PLISM	NO. PLILG	NO. FORTRAN
		EDIT	PAGE	FORTEX	PLISM	PLILG	FORTRAN						
R43	86	1:36 1:47 1:16 1:21 1:35 1:44 1:41 1:44 1:29 1:45		7:54 7:29 7:35 8:01 7:46 7:26	7:01	14:59 14:53 15:35	7:50	10	0	6	1	4	1
R44	104	1:40 1:52 2:20 2:09 2:02 1:54 2:07		11:18 10:28 10:48 9:55 10:17 10:02	5:51	19:53 20:00 20:08 20:05 19:13 20:03	2:24	7	0	6	1	7	1
R45	146			15:27 14:00 14:36 13:47 13:00	9:40	18:54 28:16 28:14 22:13 28:07 28:15 28:19 28:46 28:54 28:45 28:23 26:48 27:06	3:22 3:49	0	0	5	1	14	2
R46	148			15:39 13:37 13:55 13:01		21:02	3:32 3:40	0	0	4	2	14	2

Table C1 (continued)

RUN NUMBER	CP LOAD FACTOR	RESPONSE TIME						NO. EDIT	NO. PAGE	NO. FORTEX	NO. PLISM	NO. PLILG	NO. FORTRAN
		EDIT	PAGE	FORTEX	PLISM	PLILG	FORTRAN						
R47	68	1:08 1:11 1:17 1:21 1:05 :42 :42 1:03 :37 :33 1:08 1:20		5:12 5:19 5:06 5:14 5:25	4:14	11:20	1:20	13	0	6	1	1	1
R51	68	1:08 :43 :12 1:00 1:07 :46 :43 :51 :22 1:16 1:10 1:02 :49 1:46		4:24 5:11 7:10 5:45 5:17	7:09		1:32	15	0	5	1	1	1
R52				15:40 10:11 14:40 14:11	7:06		3:09	0	2	5	1	14	1
R53	190			10:08 10:05 6:59 7:01 11:10	12:38		3:10	0	1	5	1	15	1
	174												

Table C1 (continued)

RUN NUMBER	CP LOAD FACTOR	RESPONSE TIME						NO. EDIT	NO. PAGE	NO. FORTEX	NO. PLISM	NO. PLILG	NO. FORTRAN
		EDIT	PAGE	FORTEX	PLISM	PLILG	FORTRAN						
R54	68	1:03		5:14	5:25	10:57	1:14	15	0	5	1	1	1
		:37		5:28									
		:35		5:12									
		:59		5:39									
		1:08		4:53									
		:56											
		:26											
		:52											
		:34											
		1:09											
		:40											
		:53											
		:35											
		:46											
		:47											

Table C2: CP Mean Response Time for Each Script
(in minutes, Equation 8, section 4)

RUN SCRIPT	R21	R22	R23	R24	R25	R31	R32	R33	R34	R35	R41
EDIT	-	-	-	-	-	.92	1.43	1.33	.81	.85	.90
PAGE	-	-	-	-	-	-	-	-	-	-	-
FORTX	-	-	-	-	-	-	-	-	-	-	5.70
PLISM	-	-	-	-	-	8.52	9.78	13.71	14.42	15.30	3.20
PLILG	-	-	4.8	6.4	5.6	22.53	28.37	28.35	40.58	-	12.52
FORTAN	.25	-	.30	.80	.88	2.73	2.90	4.74	5.17	5.54	1.48
CP Load Factor	32	32	48	60	64	110	86	130	174	218	66
Combined											
Load Factor	28		40	46	55	104	68	86	104	128	

RUN SCRIPT	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT	.94	1.60	2.01	-	-	1.01	.92	-	-	.80
PAGE	-	-	-	-	-	-	-	-	-	-
FORTX	5.09	7.69	10.47	14.17	14.05	5.25	5.55	13.68	9.08	5.29
PLISM	3.63	7.00	5.85	9.67	-	4.23	7.15	7.10	12.63	5.42
PLILG	11.43	15.15	19.89	27.00	21.03	11.33	-	-	-	10.95
FORTAN	1.27	7.83	2.40	3.59	3.60	1.33	1.53	3.15	3.15	1.23
CP Load Factor	68	86	104	146	148	68	68	190	174	68

Table C3: CP Mean Calculated Throughput for Each Script (Equation 9, section 4)
(Similar to measured throughput/terminal/min)

RUN SCRIPT	R21	R22	R23	R24	R25	R31	R32	R33	R34	R35	R41
EDIT	-	-	-	-	-	1.09	.70	.75	1.23	1.18	1.11
PAGE	-	-	-	-	-	-	-	-	-	-	-
FORTEX	-	-	-	-	-	-	-	-	-	-	.18
PLISM	-	-	-	-	-	.12	.10	.07	.07	.06	.31
PLILG	-	-	.21	.156	.18	.04	.04	.04	.025	-	.08
FORTAN	4.0	-	3.3	1.24	1.14	.37	.34	.21	.19	.18	.68
Sum	4.0	-	3.51	1.41	1.32	1.62	1.18	1.07	1.51	1.42	2.36
Total Calc. Throughput	4.0*	-	3.51*	2.91*	1.32*	7.47	10.48	8.84	11.04	8.16	17.51
Subtotal Throughput**	4.0	-	3.51	2.91	1.32	3.11	2.08	1.34	1.20	1.08	

RUN SCRIPT	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT	1.06	.62	.50	-	-	.99	1.09	-	-	1.25
PAGE	-	-	-	-	-	-	-	-	-	-
FORTEX	.20	.13	.10	.07	.07	.19	.18	.07	.11	.19
PLISM	.28	.14	.17	.10	-	.24	.14	.14	.08	.18
PLILG	.09	.07	.05	.04	.05	.09	-	-	-	.09
FORTAN	.79	.13	.42	.28	.28	.75	.65	.32	.32	.81
Sum	2.42	1.09	1.24	.49	.40	2.35	2.13	.53	.51	2.52
Total Calc. Throughput	16.14	7.46	4.99	1.53	.89	13.91	16.95	.74	.95	20.78

*Unrealistically low because so few scripts had measured response times.

**Subtotal for FORTRAN, PLISM and PLILG scripts for comparison with TSS total throughput.

Table C4: CP Measured Terminal Throughput (in Job completions /min/terminal)

RUN SCRIPT	R31	R32	R33	R34	R35	R41	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT	.35	.362	.406	.425	.412	.468	.352	.375	.381	.075	-	.409	.460	-	-	.462
FORTAN	.289	.275	.194	.176	.126	.583	.625	.560	.375	.208	-	.611	.666	.290	.214	.59
FORTEX	-	-	-	-	-	.172	.166	.125	.090	-	-	.133	.133	.0735	.085	.145
PLISM	.102	.099	.053	.066	.052	.250	.208	.187	.166	.125	-	.167	.111	.117	-	.18
PLILG	.032	.041	.060	.040	-	.055	.125	.062	.041	.042	-	.055	-	-	-	-
PAGE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CP Load																
Factor	110	86	130	174	218	66	68	86	104	144	152	68	68	190	174	68
Run																
Duration (minutes)	39	37	33	25	19	36	24	16	24	24	25	18	9	17	14	22

Table C5: CP Total Measured Throughput per Script (in Job completion/min)

RUN SCRIPT	R31	R32	R33	R34	R35	R41	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT	1.38	4.35	4.06	3.40	2.47	6.52	4.58	3.75	2.67	3.75	-	4.5	6.44	-	-	5.09
FORTAN	2.02	1.65	.969	.88	.63	.583	.625	.56	.38	.416	-	.61	.67	.29	-	.59
FORTEX	-	-	-	-	-	.861	1.00	.75	.54	-	-	.67	.67	.29	.43	.73
PLISM	.307	.297	.212	.20	.157	.250	.21	.19	.17	.13	-	.167	.11	.12	-	.18
PLILG	.128	.082	.060	.08	-	.055	.13	.19	.25	.54	-	.055	-	-	-	-
PAGE	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Total Through- put	3.84	6.38	5.30	4.56	3.26	8.27	6.55	5.44	4.01	4.84	0	6.00	7.89	.70	.43	6.59
Subtotal through- put*	2.46	1.93	1.24	1.16	.79	?										

*Subtotal for FORTRAN, PLISM, and PLILG scripts for comparison with TSS total throughputs

Table C6 : CP Minimum Response Times (Minutes)
 (Used in Place of Stand-Alone Response Times in
 calculation of Effective Progress Rates.)

Source \ Script	EDIT	FORTRAN	FORTEX	PLISM	PLILG	PAGE
Haines & Porteifield's Thesis [2]	-	0.10	-	0.457	1.016	-
Terminal Probe Tests [14]	0.033 (2 sec)	0.116	0.38	0.483	1.15	6.27
Minimum during benchmark runs	0.5	0.88	3.6	2.1	8.76	-
Best Minimum Response Times	0.033	0.10	0.38	0.457	1.016	6.27
MTS (for comparison) [3]	2.84	0.18	0.28*	1.06	1.64	

*Scale by 100 from Hinson's thesis [3] because different multipliers were used.

Table C7 CP Effective Progress Rates +

RUN SCRIPT	R21	R22	R23	R24	R25	R31	R32	R33	R34	R35	R41
EDIT*	-	-	-	-	-	.036	.023	.025	.041	.039	.037
PAGE*	0	0	0	0	0	0	0	-	-	-	0
FORTEX	-	-	-	-	-	-	-	-	-	-	.067
PLISM	-	-	-	-	-	.054	.047	.033	.032	.03	.143
PLILG	-	-	.212	.159	.181	.045	.036	.036	.025	?	.081
FORTRAN	.40	-	.33	.125	.114	.037	.034	.021	.019	.018	.068
Sum+	.40	-	.542	.284	.295	.112	.14	.115	.117	.087	.396
Total											
Progress											
Rates+	.40	-	.54	.53	.30	.745	.659	.527	.569	.414	1.15
Above											
realistic	x		x	x	x	x	x	x	x	x	✓

RUN SCRIPT	R42	R43	R44	R45	R46	R47	R51	R52	R53	R54
EDIT*	.035	.021	.016	0	0	.033	.036	0	0	.041
PAGE*	0	0	0	0	0	0	0	0	0	0
FORTEX	.075	.049	.036	.027	.027	.072	.068	.028	.042	.072
PLISM	.126	.065	.078	.047	-	.108	.064	.064	.063	.084
PLILG	.089	.067	.051	.038	.048	.09	-	-	-	.093
FORTRAN	.079	.013	.042	.028	.028	.075	.065	.032	.032	.081
Sum+	.404	.215	.223	.140	.103	.378	.233**	.124**	.137**	.371
Total										
Progress										
Rates+	1.20	.85	.805	.816	.836	1.13	1.01	.236**	.305**	1.23
Above										
realistic	✓	✓	✓	✓	?	✓	?	x	x	✓

+ (minimum, or stand-alone, response time (TableC6) divided by mean response time for script i (TableC2)).

*Zero indicates no EDIT or PAGE scripts in benchmark.

**Unrealistically low because PLILG did not complete even once.

+Note the sums and the total progress rates are quite erroneous for runs R21 to R25 because response times for EDIT, FORTEX and PLISM are not available. Also the sums and total progress rates are low for runs R31 to R35 because the response time for FORTEX is not available. Runs R45, R46, R52 and R53 are ok because no EDIT scripts were run.

Table C8 CP Software Monitor Results Summary

Run	R11	R12	R13	R21	R22	R23	R24	R25	R31	R32
Average Problem Time %	51	32	64	82	86	57	46	51	16	10.5
Pages-Read (per minute)	10	23	6	9	6.5	14.9	18.6	14.8	26	32

Run	R33	R34	R35	R41	R42	R43	R44	R45	R46	R47
Average Problem Time %	7.5	8	7	39	42	38	30	25	24	36
Pages-Read (per minute)	34	32	32	19	23.5	17	20	22	21	20

Table C9: CP Software Monitor Results - Test 3

Memory Run	256k Bytes of Memory					512k	768k
	R31	R32	R33	R34	R35	R36	R37
a. CPU UTILIZATION	15.8 1.6	10.4 1.12	7.3 0.11	7.5 1.82	6.8 1.0	30.7 0.82	19.8 2.34
b. WAIT TIME	64.0 1.2	68.0 .9	72.4 0.36	72.8 1.76	73.4 1.60	49.4 2.46	63.1 3.21
c. CP TIME	15.2 0.14	16.6 .32	15.2 .36	14.5 .12	14.4 .50	15.7 1.47	13.5 1.01
d. OVERHEAD TIME	1.16	5.1 0.20	5.2 .04	5.1 .17	5.4 .10	4.1 .29	3.6 .12
e. PAGES READ/SEC	25.5 2.0	31.8 1.8	33.3 .51	31.5 2.0	31.8 1.05	10.9 2.33	10.2 53
f. PAGES SWAPPED/SEC	21.3 1.65	26.2 1.44	29.0 .52	28.6 2.19	28.5 .10	10.0 2.13	9.5 .53
g. PAGES STOLEN/SEC	20.8 1.70	26.3 2.00	29.1 .60	27.7 2.36	28.4 .65	2.83 1.23	3.6 .61
h. FORTRAN COMPILE RESPONSE TIME	2:44 :18	2:56 :48	5:00 :24	5:46 :29	7:27 2:59	—	—
i. PLISM COMPILE RESPONSE TIME	8:40 :28	9:47 :71	15:19 :06	16:07 :66	15:10 0	—	—
j. PLILG COMPILE RESPONSE TIME	22:51 :12	30:25 :71	—	—	—	—	—
k. AVERAGE THROUGHPUT PLILG (COMPILES/MIN)	.043	.033	.027	.023	.011	.072	.048
l. AVERAGE THROUGHPUT PLISM (COMPILES/MIN)	.111	.098	.067	.062	—	.226	.147
m. AVERAGE THROUGHPUT FORTRAN (COMPILES/MIN)	.338	.319	.192	.174	.139	.584	.564
n. AVERAGE THROUGHPUT FORTEX (Blips per min)	.781	.743	.424	.360	.437	1.15	1.01

Notes:

- 1) With 1 core box (256k bytes), 38 pages are usable.
- 2) In rows a through d, the first number is a percent, and the second number is the standard deviation.
- 3) In rows e through g, the first number is the number of pages per second, and the second number is the standard deviation.
- 4) In rows h through j the first number is the response time in minutes and seconds, and the second number is the standard deviation in seconds.
- 5) In rows k through m average throughput is given in number of compiles per minute for each program type. In row n, one blip is equal to 2.4 seconds of CPU time.

Table C10 Individual Terminal Response Times
For Determining Effect of Memory Size

Part a: FORTRAN (in minutes)

Run	R31	R36	R37
Compile #	One Core Box	Two Core Boxes	Three Core Boxes
1	2.37	1.41	2.03
2	2.52	1.55	2.00
3	2.24	1.25	1.09
4	2.25	1.35	1.45
5	2.41	1.28	2.08
6	2.56	1.36	1.34
7	2.30	1.43	2.08
8	2.28	1.23	1.50
9	2.48	1.43	1.12
10	2.34	1.50	1.51
11	2.27	1.43	2.08
12	3.15	1.28	1.42
13	1.21	1.45	
14	2.28	1.32	
15	2.59	1.29	
16	2.40	1.48	
17	2.46	1.27	
18	2.26	1.39	
19	2.37	1.43	
20	2.36	1.25	
Average (Minutes)	2.35	1.36	1.48

Part b: PLISM (in minutes)

1	9.18	3.55	5.56
2	8.55	4.03	7.43
3	8.38	3.51	
4	7.52	4.26	
5	9.10	4.22	
6	8.20	3.50	
7	8.13	4.18	
8	9.25	4.07	
9	8.49		
Average (Minutes)	8.45	4.06	6.34

Part c: PLILG (in minutes)

1	22.31	13.52	3.08
2	22.54	13.48	
3	22.59		
4	23.01		
Average (Minutes)	22.51	13.50	3.08*

*Very suspicious because the test run was so short.

Appendix D

TSS INTERMEDIATE RESULTS

This appendix presents the intermediate results from the TSS terminal tests that were used to prepare the graphs in Section 6.3. The intermediate results were obtained in the same way that the CP results were obtained.

Table D1 presents the TSS mean response times for all terminals running each of the scripts. Table D2 presents mean measured throughputs for each terminal in job completions per minute as determined by Equation 1 in Section 4.2. These results are the same as the mean measured throughputs per script but are not the same as the total throughputs per script as shown in Table D4. Table D3 is the equivalent of Table D2 except it contains the calculated throughputs as determined by Equation 9 in Section 4.2. The total calculated throughputs per script are shown in Table D5. The last lines of Tables D4 and D5 represent the total measured and calculated throughputs as determined by Equations 5 and 10, respectively.

Table D1: TSS Mean Response Times for each Script (minutes)

Run Script	R21	R23	R24	R25	R31	R32	R33	R34	R35
FORTAN	.30	.72	1.25	1.50	3.3	2.7	3.5	3.9	4.4
PLISM	—	—	—	—	15.8	13.1	17.5	19.0	21.0
PLILG	1.9	4.3	8.0	9.3	21.8	—	17.5	25.5	25.3
CP Load Factor	32	48	60	64	110	86	130	174	218
Combined Load Factor	28	40	46	55	104	68	86	104	122

Table D2: TSS Measured Terminal Throughput (Job Completion/min/terminal)(Equation 1)

FORTAN	2.95	1.25	.53	.59	.315	.296	.244	.222	.195
PLISM	—	—	—	—	.077	.046	.037	.035	.034
PLILG	.48	.29	.17	.12	.057	.046	.037	.035	.034

Table D3: TSS Calculated Terminal Throughput (Job completion/min/terminal)(Equation 9)

FORTAN	3.30	1.39	.8	.67	.30	.37	.29	.26	.23
PLISM	—	—	—	—	.063	.076	.057	.053	.048
PLILG	.53	.23	.13	.107	.046	—	.057	.039	.039

Table D4: TSS Total Measured Throughput per Script (Job Completions/min)($NT_i \times TP_i^m$)

FORTAN	2.95	1.25	1.56	1.77	2.20	1.48	1.27	1.11	.97
PLISM	—	—	—	—	.23	.14	.11	.105	.102
PLILG	.48	.29	.17	.12	.228	.092	.074	.07	.068
Total*	3.43	1.54	1.73	1.89	2.66	1.71	1.45	1.29	1.14

*Total determined by $\sum_{i=1}^6 NT_i \times TP_i^m$, Equation 5

Table D5: TSS Total Calculated Throughput per Script (Job Completion/min) ($NT_i * TP_i^c$)

Run Script	R21	R23	R24	R25	R31	R32	R33	R34	R35
FORTAN	3.30	1.39	2.4	2.0	2.10	1.85	1.45	1.3	1.15
PLISM	-	-	-	-	.19	.23	.17	.16	.15
PLILG	.53	.23	.13	.107	.264	-	.114	.078	.078
Total	3.83	1.62	2.53	2.11	2.55	2.08	1.73	1.54	1.38

Appendix E

MTS INTERMEDIATE RESULTS

This appendix presents the intermediate results from the MTS terminal tests that were used to prepare the figures in Section 6.5. The intermediate results were obtained in the same way that the CP results were obtained. Since Hinson made a good presentation of the intermediate results of the MTS tests in his thesis [3] these results, as shown in Table E1, were the basis for the results in this appendix.

Table E2 presents the same results as Table E1 but with several corrections to make them compatible with this project. The EDIT response times have all been reduced by 2 minutes which represents the time required to print the extra 17-1/2 lines that MTS editor produced. (See discussion in Subsection 2.2.1.) In most cases the original EDIT response times were plotted but the corrected response times were used in determining the total calculated and measured throughputs. The effective progress rates were calculated using the original EDIT response time because the corrected results would have produced unrealistically high progress rates.

The FORTEX results are corrected by multiplying the response times by 100 because the MTS tests were performed with an old version of FORTEX script that used 10,000 loops before printing 11 characters instead of the 1,000,000 used in the other tests. This is probably a little unfair to MTS since the printing time was 0.75 seconds (11 characters at 14.7 characters per second) of the 1.32 seconds minimum response time, and this does not consider the overhead of initiate the program execution, calling the clock

and printing the time an extra 99 times every 2.2 minutes.

The PLILG and the PLISM response times are also corrected for Run R63 in Table E2. Run R63 was subdivided into two runs, one representing the first few minutes of R63 when the response times were good and the other representing the rest of R63 when the response times were very poor, i.e., 4 times longer than previously measured. The sub-runs are referred to as R63a and R63b.

The analysis that lead to the separation of Run R63 is shown in Figure D1. It shows an amazing consistency of the response times within the newly defined runs R63a and R63b as well as a large difference between the response times.

Table E3 shows the times that were extracted from Tables E1 and E2 for plotting the MTS terminal response graphs. Also shown are the load factors and the number of terminals assigned to each script.

Table E4 shows the MTS measured throughput per script which was determined from the last column of Table E2 by NT_i times TP_i , where NT_i is the number of terminals running script i and TP_i is the mean throughput per terminal per minute from Table E2. The total measured throughput, with the corrected throughput for EDIT, is also shown in Table E4. (Also the total throughput with the actual EDIT throughput is shown but it was not used in the analysis.

Table E5 shows the stand-alone and light load terminal response times for each script. The light load was 5 EDIT, 1 FORTRAN, 3 FORTEX and 3 PLILG for a MTS Load factor of 54. These times were used to determine the

the effective progress rates.

The effective progress rates are shown in Table E6 which were determined by dividing the minimum response times from Table E5 by the mean response times from Tables E1 and E2 - which are also shown in Table E6. The minimum response time for EDIT was taken from the tests rather than Table E5. The total effective progress rates, as well as the EDIT subtotals and the non-EDIT subtotals, are also shown in Table E6.

Table E1: MTS Original Response Times and Throughputs*

SCRIPT	TEST NO	RESPONSE IN MIN.				THROUGHPUT IN COMPLETIONS/ MIN/TERMINAL
		MEAN	STD DEV	HIGH VALUE	LOW VALUE	
EDIT	1	2.68	0.26	2.95	2.18	0.29
	2	2.97	0.20	3.60	2.51	0.24
	3	3.43	0.33	4.54	2.97	0.22
	4	2.96	0.25	3.59	2.50	0.27
	5	3.21	0.58	4.74	2.48	0.25
	6	3.26	0.48	4.45	2.51	0.22
	7	3.36	0.46	4.62	2.91	0.23
FORTEX	1	0.0220	0	0.022	0.022	15.15
	2	0.0239	0.0001	0.0240	0.0238	10.05
	3	0.0300	0.0002	0.0302	0.0275	8.37
	4	0.0233	0.0003	0.0236	0.0230	13.80
	5	0.0234	0.0001	0.0235	0.0233	14.35
	6	0.0250	0.0007	0.0254	0.0241	6.65
	7	- -	- -	- -	- -	- -
PAGE	1	- -	- -	- -	- -	- -
	2	- -	- -	- -	- -	- -
	3	- -	- -	- -	- -	- -
	4	- -	- -	- -	- -	- -
	5	0.97	0.40	1.49	0.22	0.31
	6	- -	- -	- -	- -	- -
	7	- -	- -	- -	- -	- -
PLILG	1	2.86	0	2.86	2.86	0.31
	2	3.97	0.41	4.47	3.26	0.17
	3	21.00	14.54	37.22	6.71	0.025
	4	2.94	0.40	3.54	2.32	0.32
	5	6.13	1.08	8.01	4.83	0.18
	6	6.85	1.48	9.43	5.73	0.063
	7	6.70	1.12	9.08	4.77	0.13
PLISM	1	- -	- -	- -	- -	- -
	2	2.36	0.23	2.63	2.16	0.29
	3	7.24	4.40	11.67	2.94	0.11
	4	1.71	0.20	1.85	1.57	0.22
	5	3.55	0.94	4.75	2.52	0.17
	6	3.61	0.52	4.39	2.99	0.19
	7	3.53	1.26	4.45	2.10	0.24
FORTRAN	1	0.37	0.21	0.59	0.18	0.57
	2	0.41	0.21	0.82	0.20	1.49
	3	1.16	1.03	3.81	0.25	0.31
	4	0.36	0.26	0.89	0.20	1.67
	5	0.60	0.28	1.19	0.22	1.86
	6	0.67	0.39	2.70	0.20	1.06
	7	0.58	0.34	1.74	0.20	1.14

*Obtained from Hinson's thesis [3].

Table E2: MTS Corrected Response and Throughput

SCRIPT	TEST NO	RESPONSE IN MIN.				MEASURED THROUGHPUT IN COMPLETIONS/ MIN/TERMINAL
		MEAN	STD DEV	HIGH VALUE	LOW VALUE	
EDIT	R61	0.68*	-	0.95	0.18	1.14*
	R62	0.97	-	1.60	0.51	.73
	R63	1.43	-	2.54	0.97	.53
	R64	0.96	-	1.59	0.50	.83
	R65	1.21	-	2.74	0.48	.66
	R66	1.26	-	2.45	0.51	.57
	R67	1.36	-	2.62	0.91	.57
FORTEX x 100	R61	2.20	-	2.2	2.2	.151
	R62	2.39	0.01	2.40	2.38	.100
	R63	3.00	0.02	3.02	2.75	.083
	R64	2.33	0.03	2.36	2.30	.138
	R65	2.34	0.01	2.35	2.33	.143
	R66	2.50	0.07	2.54	2.41	.066
	R67	-	-	-	-	-
PAGE	R61	-	-	-	-	-
	R62	-	-	-	-	-
	R63	-	-	-	-	-
	R64	-	-	-	-	-
	R65	0.97	0.40	1.49	0.22	0.31
	R66	-	-	-	-	-
	R67	-	-	-	-	-
PLILG	R61	2.86	0	2.86	2.86	0.31
	R62	3.97	0.41	4.47	3.26	0.17
	R63a	8.5	4.4	12.0	6.71	} 0.025
	R63b	35.4	7.94	37.22	28.3	
	R64	2.94	0.40	3.54	2.32	0.32
	R65	6.13	1.08	8.01	4.83	0.18
	R66	6.85	1.48	9.43	5.73	0.063
	R67	6.70	1.12	9.08	4.77	0.13
PLISM	R61	-	-	-	-	-
	R62	2.36	0.23	2.63	2.16	0.29
	R63a	3.2	.39	3.5	2.94	} 0.11
	R63b	11.0	1.31	11.67	10.0	
	R64	1.71	0.20	1.85	1.57	0.22
	R65	3.55	0.94	4.75	2.52	0.17
	R66	3.61	0.52	4.39	2.99	0.19
	R67	3.53	1.26	4.45	2.10	0.24
FORTRAN	R61	0.37	0.21	0.59	0.18	0.57
	R62	0.41	0.21	0.82	0.20	1.49
	R63	1.16	1.03	3.81	0.25	0.31
	R64	0.36	0.26	0.89	0.20	1.67
	R65	0.60	0.28	1.19	0.22	1.86
	R66	0.67	0.39	2.70	0.20	1.06
	R67	0.58	0.34	1.74	0.20	1.14

* 2 minutes subtracted from results in Table E1,
i.e., $.29 \times 2.68/0.68 = 1.14$

Figure E1: Time Analysis
of MTS Run R63

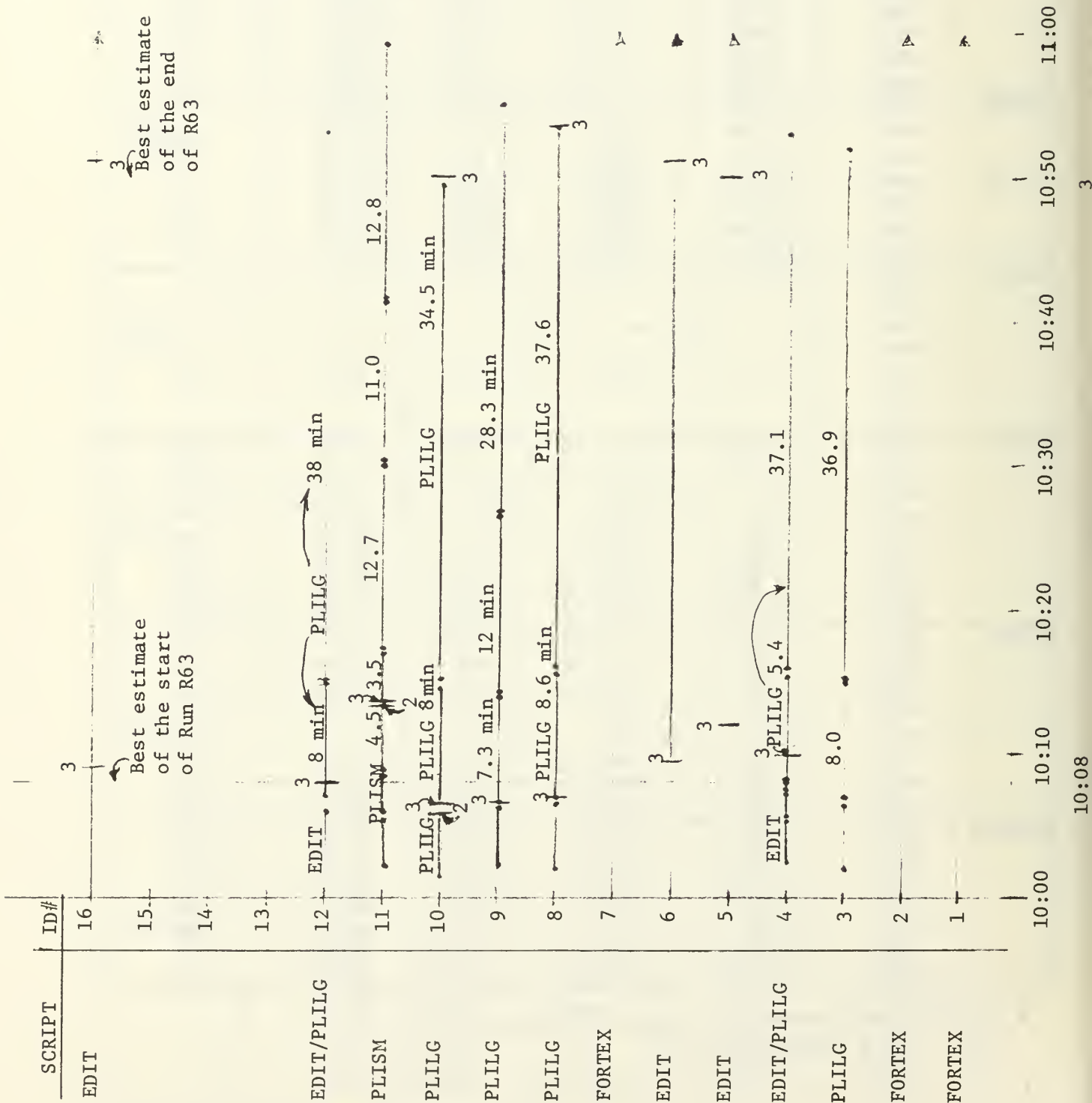


Figure E1 (cont'd): Time Analysis of
MTS Run R63

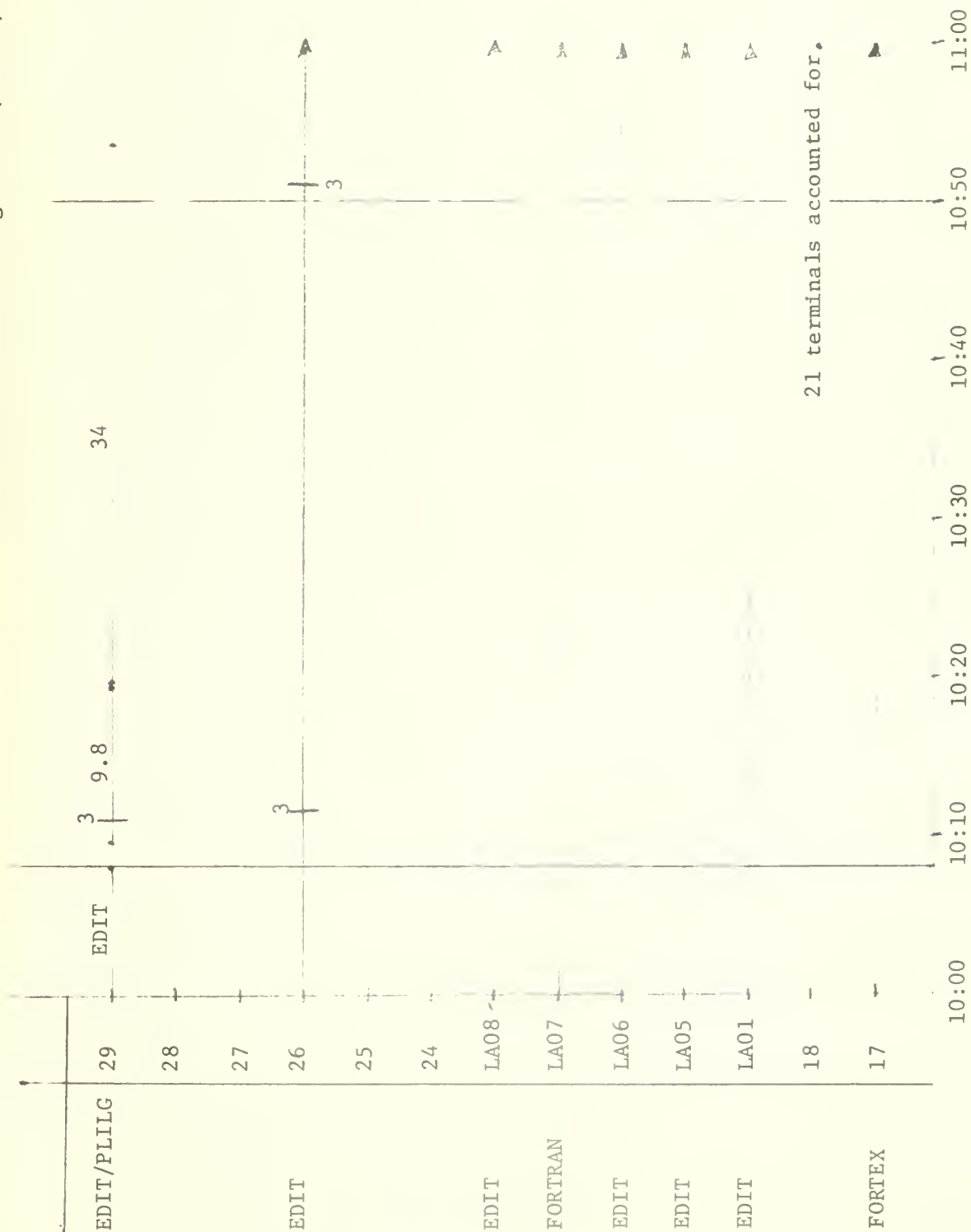


Table E3: MTS Mean Terminal Response Times Used in Graphs (in Minutes)

RUN #	LOAD FACTOR		EDIT R.T.	FORTEX R.T.	PLISM R.T.	PLILG R.T.	FORTRAN R.T.	# EDIT	# FORTEX	# PAGE	# PLILG	# PLISM	# FORTRAN
	CP	MTS	COMB										
R61	48	44	39	2.68	2.20	2.86	.37	11	3	0	1	0	1
R62	86	84	77	2.39	2.39	3.97	.41	13	4	0	4	1	1
R63a	104	142	98	3.43	3.00	8.5	1.16	10	4	0	7	1	1
R63b	104	142	98	3.43	3.00	35.4	1.16	10	4	0	7	1	1
R64	86	84	72	2.96	2.33	2.94	.36	8	4	0	4	1	1
R65	114	98	102	3.21	2.34	6.13	.60	8	3	2	4	3	3
R66	110	132	101	3.26	2.50	6.85	.67	5	6	0	4	3	5
R67	98	113	80	3.36	-	6.70	.58	12	0	0	7	1	3

Table E4: MTS Measured Throughput per Script*

Run Script	R61	R62	R63	R64	R65	R66	R67
EDIT (actual; Table E1)	3.19	3.12	2.20	2.16	2.00	1.10	2.76
FORTTRAN	.57	1.49	.31	1.67	5.58	5.30	3.42
FORTTRAN (Corrected)	.45	.40	.34	.55	.43	.40	-
PLISM	-	.29	.11	.22	.51	.57	.24
PLILG	.31	.68	.18	1.28	.72	.25	.91
TOTAL	4.42	5.98	3.14	5.88	9.24	7.62	7.33
EDIT (corrected; Table E2)	12.5	9.5	5.3	6.6	5.3	2.9	6.8
Subtotal without EDIT	1.33	2.86	.94	3.92	7.24	6.52	4.57
Total with corrected EDIT	13.8	12.4	6.2	10.5	12.5	9.4	11.4
Combined Load Factor	39	77	98	72	102	101	80
MTS Load Factor	44	84	142	84	97	132	113

Table E5: MTS Minimum Terminal Response Time

SCRIPT	SINGLE JOB IN SYSTEM	12 JOBS IN SYSTEM
EDIT	2.84**	2.85
FORTEX	-	0.28
FORTTRAN	0.18	0.18
PAGE	0.18	-
PLILG	1.64	1.93
PLISM	1.06	-

*Determined from last column of Table E2 by $NT_i \times \overline{TP}_i$ where NT_i is the number of terminal running script i and $\overline{TP}_i$ is the mean throughput per terminal per minute.

**Too high; the lowest value in the tests was 2.18 minutes.

Table E6: MTS Effective Progress Rates

Run Mean Response Times - Tables E1 and E2	R61	R62	R63a	R63b	R64	R65	R66	R67	Stand-alone
EDIT	2.68	2.97	3.43		2.96	3.21	3.26	3.36	2.18
FORTAN	0.37	0.41	1.16		0.36	0.60	0.67	0.58	0.18
FORTX	2.20	2.39	3.00		2.33	2.34	2.50	-	0.28
PLISM	-	2.36	3.2	11.0	1.71	3.55	3.61	3.53	1.06
PLILG	2.86	3.97	8.5	35.4	2.94	6.13	6.85	6.70	1.64
Mean Effective Progress Rates									
EDIT	0.813	0.735	0.635		0.737	0.68	0.67	0.65	1
FORTAN	0.49	0.44	0.155		0.50	0.30	0.27	0.31	1
FORTX	0.127	0.117	0.093		0.12	0.12	0.112	-	1
PLISM	-	0.45	0.33	0.096	0.62	0.30	0.294	0.30	1
PLILG	0.574	0.413	0.19	0.045	0.56	0.268	0.24	0.245	1
Total*	10.39	12.56	8.56	7.29	9.74	8.77	7.23	10.75	
Subtotal for EDITs	8.95	9.55	6.35	6.35	5.90	5.44	3.35	7.80	
Subtotal for Non-EDITs	1.44	3.01	2.21	0.94	3.84	3.33	3.88	2.95	

*Total Effective progress rates = $\sum_{i=1}^5 NT_i \times EPR_i$, where NT_i and EPR_i are the number of terminals running script i and its mean effective progress rate, respectively.

Appendix F

BATCH INTERMEDIATE RESULTS

The intermediate data from a late version of the MTS batch benchmarking runs is shown in Figure F1. The data includes the starting, stopping, elapsed and CPU times and the lines of printed output for each job. Also shown are the order of job initiation and the total elapsed time. These tests on jobstream A and B were performed in September 1971.

The intermediate data for OS benchmarking tests performed in November of 1969 under OS/MVT version 15/16 are shown in Table F2. The data includes the total turnaround times, the times in which at least 3 jobs were in the system and the multiprogramming times.

A summary of above MTS and OS results as well as some other tests on MTS and TSS is shown in Table F3 which summarizes the total turnaround time for each jobstream and configuration.

Table F1: MTS Batch Benchmarking Results - Sept/71Part 1 Jobstream A

Jobs \ Times	Start time (ON) (hours:min:sec)	Stop Time (OFF) (hours:min:sec)	Elapsed Time (seconds)	CPU Time (seconds)	Lines Printed
SYS001	5:42:07	5:42:27	20.8	15.0	189
SYS003	5:42:34	5:42:48	14.0	5.16*	476
SYS002	5:42:31	5:43:10	38.7	19.1	1243
SYS005	5:43:17	5:43:53	35.9	10.9	300
SYS011	5:45:12	5:46:08	55.9	10.6	278
SYS004	5:43:05	5:44:18	72.4	21.7	602
SYS008	5:44:12	5:45:16	64.1	14.4	409
SYS010	5:44:49	5:45:52	63.0	9.39	658
SYS007	5:43:57	5:45:09	71.3	19.3	1213
SYS006	5:43:32	5:45:18	106.0	17.7	1227
SYS015	5:46:12	5:51:23	311.0	93.0	74
SYS014	5:46:00	5:51:37	366.4	93.4	74
SYS012	5:45:21	5:46:33	71.5	18.4	1232
SYS009	5:44:41	5:46:45	123.8	21.3	1178
SYS013	5:45:38	5:56:05	626.7	364.1	604

Order of job starts: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
 Total Turnaround Time = 5:56:05 - 5:42:07 = 13:58 = 13.97 Minutes

Part 2: Jobstream B

Jobs \ Times	Start Time (ON) (hours:min:sec)	Start Time (OFF) (hours:min:sec)	Elapsed Time (seconds)	CPU Time (seconds)	Lines Printed
SYS001	-	-	~21.	-	-
SYS002	13:03:33	13:06:26	173.0	19.4	1029
SYS004	13:04:06	13:07:08	182.2	21.0	69
SYS008	13:06:13	13:07:25	71.7	12.2	183
SYS007	13:06:06	13:06:43	42.5	18.8	1029
SYS011	13:06:58	13:08:11	72.5	10.2	57
SYS006	13:04:33	13:06:47	135.0	15.6	973
SYS009	13:06:35	13:07:44	68.8	20.5	716
SYS015	13:07:51	13:10:43	171.6	91.5	59
SYS014	13:07:29	13:10:46	196.9	91.6	59
SYS010	13:06:48	13:07:44	55.6	8.95	550
SYS012	13:07:12	13:08:13	60.6	15.3	973
SYS013	13:08:01	13:15:24	443.2	363.3	251
SYS003	13:03:35	13:05:52	136.7	5.11**	459
SYS005	13:04:18	13:06:10	112.0	10.6	118

Order by Start time: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 13
 Total Turnaround Time = 15:24 - 03:33 = 11:51 plus 21 sec = 12.2 minutes

* CPU time limit exceeded on SYS003

** Normal termination

Table F2: OS/MVT Batch Benchmarking Results

Configuration	Job-stream	Total Turnaround Time (minutes)	Third-in to Third-out Time* (minutes)	Multi-programming Time** (minutes)
3 core boxes	A	22	10	18
	B	22	12	17
	C	26	15	16
2 core boxes	A	24	13	20
	B	22 ⁺	11	17
	C	27 ⁺	16	7

Table F3: Summary Batch Total Turnaround Times (minutes)

System	Configurations	Jobstream		
		A	B	C
OS/MVT	3 core boxes	22	22	26
	2 core boxes	24	22 ⁺	27 ⁺
TSS	3 core boxes Dual processor	20	16	14
	3 core boxes Single processor	26	21	21 ⁺
	2 core boxes Single processor One channel Control Unit	41 ⁺	31 ⁺	27 ⁺
MTS	3 core boxes Dual processor	12.5	11.3	10.0
Summer 1970	2 core boxes Single processor One channel Control Unit			19.5
MTS Sept/71	3 core boxes Dual processor	13.97	12.2	

* Third-in until third-out time means the time in which at least 3 jobs were in the system.

** Multiprogramming time means the time in which at least 2 jobs were executing.

INITIAL DISTRIBUTION LIST

	No. Copies
Fleet Material Support Office Code 964 Mechanicsburg, Pennsylvania 17055	1
Defense Documentation Center (DDC) Cameron Station Alexandria, Virginia 22314	12
Library Naval Postgraduate School Monterey, California 93940	2
Dean of Research Naval Postgraduate School Monterey, California 93940	2
Library Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
W. R. Church Computer Center Naval Postgraduate School Monterey, California 93940	2
Office of Naval Research, Code 437 Information Systems Program Department of the Navy Arlington, Virginia 22217	2
Director, Naval Research Laboratory ATTN: Library, Code 2029 (ONRL) Washington, D. C. 20390	1
Dr. G. H. Syms, Code 53Zz Computer Science Group Naval Postgraduate School Monterey, California 93940	5
LT F. Douglas Meyer, USN Naval Destroyer School Newport, Rhode Island 02840	1

CDR Robert E. Graham, USNR Naval Reserve Training Headquarters New Orleans, Louisiana 70140	1
Mr. Mike A. Lamendola, Code 5200 Naval Electronic Laboratory Center San Diego, California 92152	1
Mr. Mike T. Alexander Computer Center Building 1075 Beal Avenue University of Michigan Ann Arbor, Michigan 48105	1
Mr. Gerald N. Cederquist Assistant Research Engineer Cooley Electronic Laboratory of the Department of Electrical and Computing Engineering University of Michigan Ann Arbor, Michigan 48105	1
Chairman, Department of Computer Science University of British Columbia Vancouver, B. C., CANADA	1
Dr. T. A. Marsland Department of Computer Science University of Alberta Edmonton 7, Alberta, CANADA	1
LCDR Elbert F. Hinson, USN 9 Legare Street Charleston, South Carolina 29401	1
LT William R. Haines, USN 38657 Rhons Wood Court Northville, Michigan 48167	1

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)		2a. REPORT SECURITY CLASSIFICATION	
Naval Postgraduate School Monterey, California 93940		UNCLASSIFIED	
3. REPORT TITLE		2b. GROUP	
Benchmarked Comparison of the TSS/360, CP/67, MTS and OS/MVT Computer Operating Systems			
4. DESCRIPTIVE NOTES (Type of report and, inclusive dates)			
Technical Report, 1973			
5. AUTHOR(S) (First name, middle initial, last name)			
Gordon H. Syms			
6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO OF REFS	
June 1973		16	
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)		
Foundation Grant	NPS-53ZZ73061A		
b. PROJECT NO.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)		
c.			
d.			
10. DISTRIBUTION STATEMENT			
Approved for public release; distribution unlimited			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
		Naval Postgraduate School	
13. ABSTRACT			
A set of terminal scripts and benchmarks have been derived for comparing the performance of time sharing and batch computer operating systems. Some of the problems encountered in designing valid benchmarks for comparing computer operating systems under both terminal and batch loads are discussed. The results of comparing TSS/360, CP/67 and MTS time sharing systems for the IBM 360/67 over a wide range of load conditions are presented. The results of comparing TSS, MTS and OS/MVT under batch loads are also presented. Serious performance degradation of the time sharing computer systems from overloading was experienced and a simple solution is suggested to prevent such degradation. The degradation was so sever as to render the performance less than that of a sequential job processor system.			

DD FORM 1473

1 NOV 65

(PAGE 1)

192

UNCLASSIFIED

Security Classification

S/N 0101-807-6811

A-31408

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Benchmarking						
Performance						
Evaluation						
Time Sharing System						
Time Sharing						
Computer Performance Evaluation						
Measurement						
Terminal Scripts						
Operating System						
Computer						
CP/67						
CP/CMS						
TSS/360						
MTS						
OS/MVT						
IBM 360/67						
Michigan Terminal System						
Load Factor						
Effective Progress Rate						
Computer Performamnce						
Batch System						

U155293



5 6853 01060478 8

U15529